Vittorio Bilò, Antonio Caruso (Eds.)

ICTCS 2016

Seventeen Italian Conference on
Theoretical Computer Science

Lecce, Italy, September 7–9, 2016.
Proceedings.

ISSN 1613-0073
CEUR Workshop Proceedings, Volume 1720 (http://ceur-ws.org/Vol-1720)

Editor's address:
Università del Salento
Dipartimento di Matematica e Fisica "Ennio De Giorgi"
Via per Arnesano, Campus Ecotekne
73100 Lecce, Italy.

antonio.caruso@unisalento.it, vittorio.bilo@unisalento.it

Preface

The Italian Conference on Theoretical Computer Science (ICTCS) is the annual conference organized by the Italian Chapter of the European Association for Theoretical Computer Science (EATCS). The scope of the meeting is fostering the cross-fertilization of ideas stemming from different areas of Theoretical Computer Science. Hence, it represents an occasion for meeting and exchanging ideas and for sharing experiences between researchers; it also provides an ideal environment where junior researchers and PhD students can meet senior researchers.

Topics of interest for ICTCS are: agents, algorithms, argumentation, automata theory, automated theorem proving, complexity theory, computational logic, computational social choice, concurrency, cryptography, discrete mathematics, distributed computing, dynamical systems, formal methods, game theory, graph theory, knowledge representation languages, model checking, process algebras, quantum computing, rewriting systems, security and trust, semantics, specification and verification, systems biology, types.

This volume contains papers presented at the 17th Italian Conference on Theoretical Computer Science (ICTCS'16), held in Lecce during September 7-9, 2016, together with abstracts of invited lectures by Giampaolo Bella (Università di Catania) and Gianluigi Greco (Università della Calabria). Two types of contributions were solicited: *regular papers* and *short communications*. The former consisted in full original papers, presenting novel results, not appeared or submitted elsewhere, while the latter encompassed extended abstracts of papers already appeared, or submitted, or to be submitted elsewhere; papers reporting on ongoing researches on which the authors wished to get feedback and possibly intended to be included in future publications; overviews of PhD-theses, research projects, and so on. We received a total of 35 submissions. Regular papers and short communications were reviewed by, respectively, three and two referees who judged them for originality, quality, correctness and consistency with the topics of the conference. Based on the referees' reports, the Program Committee decided to accept a total of 29 submissions: 15 regular papers and 14 short communications.

We would like to thank all the authors who responded to the call for papers and our invited speakers Giampaolo Bella and Gianluigi Greco.

Furthermore, we thank the members of the Program Committee and the subreferees for their excellent and qualified work.

We gratefully acknowledge sponsorship from the Univerità del Salento and its Department of Mathematics and Physics "Ennio De Giorgi", the Fondazione Puglia, and the Italian Chapter of the European Association of Theoretical Computer Science.

Lecce
September 20, 2016

Vittorio Bilò
Antonio Caruso

Conference Organization

ICTCS'16 is organized by the department of Mathematics and Physics "Ennio De Giorgi", Univeristy of Salento in cooperation with the Italian chapter of EATCS.

Conference Chairs

Vittorio Bilò	Università del Salento
Antonio Caruso	Università del Salento

Program Committee

Luca Aceto	Reykjavik University
Stefano Bistarelli	Università di Perugia
Chiara Bodei	Università di Pisa
Andrea Clementi	Università "Tor Vergata" di Roma
Rocco De Nicola	IMT - School for Advanced Studies, Lucca
Mariangiola Dezani	Università di Torino
Michele Flammini	Università di L'Aquila
Giovanni Pighizzini	Università di Milano
Geppino Pucci	Università di Padova
Paola Vocca	Università della Tuscia

Additional Reviewers

Basile, Davide	García-Pérez, Álvaro	Padovani, Luca
Bedogni, Luca	Giuliodori, Paolo	Pasquale, Francesco
	Goldwurm, Massimiliano	Pomello, Lucia
Ceccarello, Matteo		Prencipe, Giuseppe
Crafa, Silvia	Knapik, Michał	Proietti, Guido
D'Angelo, Gianlorenzo	Leucci, Stefano	Reiter, Fabian
D'Emidio, Mattia		
Della Monica, Dario	Manea, Florin	Santini, Francesco
Deng, Yuxin	Mantaci, Sabrina	Sokolova, Ana
	Massink, Mieke	Sproston, Jeremy
Fantozzi, Carlo	Merro, Massimo	Stievenart, Quentin

Fell, Alexander
Ferraioli, Diodato
Ferrari, Gianluigi
Fiorentini, Camillo
Francalanza, Adrian

Gadducci, Fabio
Galletta, Letterio

Miculan, Marino
Milazzo, Paolo
Monaco, Gianpiero
Moscardelli, Luca
Mukherjee, Manideepa

Natale, Emanuele
Nenzi, Laura

Tannock, Murray
Tiezzi, Francesco
Torres Vieira, Hugo

van Breugel, Franck

Sponsoring Institutions

Dipartimento di Matematica e Fisica 'Ennio De Giorgi"
Fondazione Puglia

Table of Contents

ICTCS 2016

Invited Speakers

Regular Papers

Short Communications

Invited Speakers

Cybersecurity's Way Forward: to get Beautiful or Invisible

Giampaolo Bella

Dipartimento di Matematica e Informatica, Università di Catania, Italy
giamp@dmi.unict.it

Abstract. People do not generally like Cybersecurity. Although they do believe it is somewhat good to have, they often cannot be bothered to go through security defences such as registrations, strong passwords' choices, PINs' long waits through the post and all the like. I do believe they are essentially right, especially if modern services are to be enjoyed on the move, while the user is hopping on the tube, or pervasively, while the user is also watching television. Sometimes users *have to* be bothered to go through such defences otherwise they will not get the service they wanted. They may then be nastily rewarded with senses of disappointment and frustration both if they opted to go on and if their pride or boredom prevented them to.
A layman at a cafe was arguing that he found Cybersecurity especially hideous when he was in a rush to get some service. Almost every researcher who looks at Cybersecurity from the socio-technical angle will agree with that layman as much as I do. This position paper outlines my view of the sole way forward for Cybersecurity: a fork in the road that takes either to Beautiful City or to Invisible City. One may of course refuse the fork and go back on the same road to Old City, where Cybersecurity often failed for a variety of reasons, including purely technical bugs and human-centred mistakes. I will postulate how I envisage Beautiful City and Invisible City to be. And do not worry you formal methodists: your help will be most appreciated also in the new cities.

1 Rationale

A modern understanding of Cybersecurity situates it in a real use scenario that sees human users approach a technology that is meant to be secure, and experience it or, more simply, just use it. This is certainly a fuller and yet more insightful understanding than the traditional one, at least because it is clear that no technology will be secure if its users keep the login passwords on sticky notes.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 1–7 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

Here comes a new breed of views of Cybersecurity that are pivoted on the users, featuring the *social*, *economic* and *legal* views. These bear a huge potential to unveil niceties that could not be spot before, such as how easy it is for the layman to learn and comply with Cybersecurity, the coexistence with a deployed-though-flawed system version, and the assistance of the law to users who are victims of real-world breaches.

These new views entail what the Technocrats may perceive as a revolution: it will not suffice to look at the technical system in all sorts of ways to make it secure as they were used to do; by contrast, Scientists will have to look at the technical system holistically with its human users, and make that larger "system" secure. Arguably, Scientists will have to collaborate with colleagues from the Humanities to account for the human element. They will still only pass a technical system on to Engineers to build, but the resulting technology will be secure and privacy-preserving when practically used.

The new views of Cybersecurity in fact attract a worldwide, interdisciplinary task force of researchers at present. These are not just Computer Scientists but also Sociologists, Psychologists, Economists and Lawyers, confirming once and for all to everyone that the topic is not a purely technical one, as someone might have believed. A number of research events have appeared to publish the new research output, notably the Workshop on Socio-Technical Aspects in Security and Trust (STAST) [1], the Workshop on the Economics of Information Security (WEIS) [2], and the Workshop on TEchnical and LEgal aspects of data pRIvacy and SEcurity (TELERISE) [3] to just mention one per view.

2 Technology Users

Humans are difficult to fully account for, let alone formalise in the way dear to Formal Methodists. In particular, the human users of a technology are far from being automata executing the perfect program that the Technocrats behind that technology had in mind:

Users may be deceived It is consolidated at least since Mitnick published his famous book on deception that humans are rather easy to be duped into making insecure actions, such as choosing poor passwords or annotating secrets in insecure places [4]. It turns out that humans may effectively be tricked into facilitating the attacker's aims.

Users may make errors There exists vast work from the Humanities studying how and why humans make errors. Norman catalogued errors either as a failure to do what the user intends (*mistakes*) or a momentary lapse when the user takes an unintended action (*slips*) [5]. For example, both types of errors might be due to the often innate quest to operate in a best-effort way.

Users may choose to counter Cybersecurity When humans feel Cybersecurity as a burden more than as a benefit, they may deliberately ignore or oppose it. For example, some companies require card-and-PIN authentication to enter their premises or record work times, but Amazon suggests that cards can be left in a public card rack near the PIN pad [6].

3 The Cybersecurity Planet

I have done some research on the social view of Cybersecurity (not yet on the others), and this position paper gives me the opportunity to summarise and review some of my findings. At the moment, I see Cybersecurity as a planet with just three cities: Old City, Beautiful City and Invisible City. There exists a road that departs from Old, then forks and takes either to Beautiful or to Invisible. I am afraid that I have not explored anything else of that planet yet. It would seem that we researchers have been given the power to put the human users of the technology that we want to be secure in any of the three cities. I speculate that we used to choose Old but we had better take the users to either Beautiful or Invisible if we really care that technology to be secure.

3.1 Old City

This is the oldest city on the Cybersecurity planet, hence technology users have lived it for a long time. Here, Cybersecurity can be particularly hard to understand, interpret and use, and it can be realised empirically that it is often vulnerable. Vulnerabilities exist despite the Technocrats' best efforts at preventing them, hence all sorts of security incidents have happened over time.

Vulnerabilities are not only purely technical as with the SSL Heartbleed and Shellshock bugs. IBM reported that "*over 95 percent of all incidents investigated recognize 'human error' as a contributing factor*" in the 2014 Cyber Security Intelligence report [7], a trend that has not substantially changed ever since. One example dating back to a couple of years ago is how a user could deliberately share a file she stored on Dropbox or on Box with other users and inadvertently disclose it to unwanted parties[8]. Even the established policy of asking users to change their passwords from time to time may falter. It was recently found out that humans often resort to simple, algorithmic changes of the previous password to build the new one, hence attackers will just have to fine-tune their brute-forcing techniques [9]. Cybersecurity often intertwines with people's safety. Last year's Chatham House Report shouts out loud to the world that "*Some nuclear facilities do not change the default passwords on their equipment*" [10].

3.2 Beautiful City

In this city, Cybersecurity is beautiful [11], and I contributed some definition on what that means. Cybersecurity is beautiful if it satisfies a triple of abstract requirements: to be a primary system feature, not to be disjoint from the system functions to be secured, and to be ambassador of a positive user experience. I am going to expand them below.

The first one is not innovative by itself as it appeals to the security-by-design principle that the system should be designed with security in mind since the beginning; this normally enhances the ease of use and at the same time the effectiveness of the security defences. For example, security experts should contribute to the design of *at least* security-sensitive services since the inception.

The second requirement insists on what even security-by-design fails to prescribe clearly, that the secure access to a service be exclusive, namely the only possible one. For example, let us think of a web site secured via HTTPS yet allowing access also via HTTP for whatever legacy or performance reason. Also, when a user connects to a remote host via SSH for the first time, the user will have to accept the host's public key on trust rather than on a viable certification system.

The third requirement is perhaps the most abstract one. I would like Cybersecurity to be nice, desirable, rewarding and, generally speaking, a somewhat positive thing to have. One way to meet this requirement could be to aim at a Cybersecurity perceived as an engaging and fun game. An episode of the Peppa Pig cartoon portrays a group of kids wanting to be part of a "*secret club*" as soon as they come to know of its existence [12]. Can we manage to upturn people's currently negative perception of Cybersecurity to match the cartoon's?

The gist of the beautiful security principle is that all three requirements be met at the same time. It would seem that the use of the web interface of WhatsApp conforms to this principle. The web client prompts the server, who then issues a passcode for the former, stores it and sends it back; the web client displays it as a QR code, which the phone client (the app) scans and sends to the server along with the chat log stored on the phone. Only if the received version matches the stored version, will the server output the chat log to the web client.

Notably, it all takes place over HTTPS except for the step whereby the passcode reaches the phone, which involves a human pointing the phone to the computer screen to scan the QR code. This is a crucial design choice: the passcode is 128 characters long, hence it would have been super tedious for the user to have to read it from the computer screen and tap it in the phone. Here, QR-code scanning conjugates usability, simplicity, security and also some beauty. I gather from random discussions that QR-code scanning normally thrills people.

3.3 Invisible City

In Invisible City, Cybersecurity is not perceived by the technology users although it is still there. The idea is that if we cannot conjugate users and security by means of beauty, then the only option left seems to make security invisible, that is to literally make it invisible to the users' perceptions. I provided various examples on how this could be achieved in practice by integrating the security defences with system functions or with other defences that the users would accept as routine [13].

My favourite example is the Iphone 5S's integration of the screen activation button with the fingerprint sensor. This idea stemmed from the observation that people were used to a stand-by display being off to preserve battery, hence to the need of pressing some button to activate it when needed. This integration combined a routine action with an important security defence, user authentication to the phone, which otherwise required a separate ceremony to insert a PIN or password.

I argue that another security ceremony that could live well in this city would be a modification of the ceremony whereby passengers currently board flights (at the gates of Old City airport). Each passenger gives the gate attendant three pieces of information: the passenger's face, his ID and his boarding pass. The attendant matches face to ID, checks the ID validity, matches the ID to the boarding pass, scans the pass in the airport system and checks that the outcome confirms the identity that is allowed on the flight currently boarding. Only if all checks succeed will the passenger be allowed to go through, otherwise he will be stopped for further scrutiny. These are a number of checks for the attendant to carry out on each passenger in a long queue, hence not surprisingly some passengers complained to have reached the wrong destination [14].

With airport security being so sensitive at present, this scenario could be easily turned into various types of threats if the passenger attempted it deliberately and without reporting it. Therefore, I suggest to completely dispose with the boarding pass. This would leave only the initial authentication checks and the final authorisation one performed on the ID, which should be an electronic one, rather than on the pass. The match between the details of the ID with those of the boarding pass would be eliminated, reducing the risk of mismatching an authenticated identity to an identity that is authorised to board the flight.

4 Formal methods

I am a formal methodist down to the bone, so it is no surprise if I believe that formal methods can help a great lot to assess Cybersecurity from the socio-technical angle, hence to build both Beautiful City and Invisible City. I have published a few contributions to this debate [15,16]. In particular, I used the Cognitive Walkthrough usability inspection method to analyse Amazon's sub-ceremonies for price-quotation, shopping and purchase of the time, observing a few weaknesses. A notable one was that a user could choose a weak login password without getting any warning, and his credit card details would be recorded and protected merely by that weak password. Although these ceremonies have changed repeatedly over time, one of the conclusions of the analysis was:

> "*Amazon should clarify that the password that a user chooses during Registration has an impact on the confidentiality of their credit card details during network traversal at time of Purchase. Hence, Amazon should encourage each user to choose a strong password.*" [17].

The value of this recommendation does not expire; it could be generalised to every service recording users' sensitive information.

5 Conclusions

This position paper demonstrates my view of Cybersecurity as a socio-technical problem, namely one that pertains to both the technology and to how users

receive and avail themselves of it. Cybersecurity is a planet featuring Old City, one in which vulnerabilities and their exploitations are also due to an insufficient account on how humans use technology. I envisaged a road departing from Old City taking to either Beautiful City or to Invisible City, where I argued that Cybersecurity is more mindful of the human element. I also described some recent example uses of formal methods to help consolidate and expand the two new cities. Those are the places where I conclude that we researchers in Cybersecurity had better "move" the technology users from Old City.

References

1. URL: Workshop on socio-technical aspects in security and trust (2016) http://stast.uni.lu/.
2. URL: Workshop on the Economics of Information Security (2016) http://weis2016.econinfosec.org/.
3. URL: Workshop on TEchnical and LEgal aspects of data pRIvacy and SEcurity (2016) – http://www.iit.cnr.it/telerise2016/.
4. Mitnick, K.D., Simon, W.L.: The Art of Deception: Controlling the Human Element of Security. John Wiley & Sons (2001)
5. Norman, D.A.: Categorization of action slips. Psychological Review **88** (1981) 1–15
6. URL: STEELMASTER Swipe Card or Badge Rack (2016) https://www.amazon.com/STEELMASTER-Swipe-Capacity-Inches-20401/dp/B002V85VWQ.
7. URL: IBM Security Services 2014 Cyber Security Intelligence Index (2016) http://media.scmagazine.com/documents/82/ibm_cyber_security_intelligenc_20450.pdf.
8. URL: Dropbox, Box users Leak Sensitive Information via Shared Links Flaw (2014) – http://techfrag.com/2014/05/08/dropbox-box-users-leak-sensitive-information-via-shared-links-flaw/.
9. URL: Frequent password changes are the enemy of security, FTC technologist says (2016) – http://arstechnica.com/security/2016/08/frequent-password-changes-are-the-enemy-of-security-ftc-technologist-says/.
10. URL: Chatham House Report (2015) https://www.chathamhouse.org/sites/files/chathamhouse/field/field_document/20151005CyberSecurityNuclearBaylonBruntLivingstoneUpdate.pdf.
11. Bella, G., Viganò, L.: Security is Beautiful. In: Proceedings of the 23rd International Workshop on Security Protocols (SPW'15). LNCS 9379, Springer (2015) 247–250
12. URL: Peppa Pig, Series 3, Episode 38, "The Secret Club" (2010) https://www.youtube.com/watch?v=QSQhScDvOao.
13. Bella, G., Christianson, B., Viganò, L.: Invisible Security. In: Proceedings of the 24th International Workshop on Security Protocols (SPW'16). LNCS Series, Springer (2016) In press.
14. URL: Ryanair passenger gets on wrong plane and flies to Sweden instead of France (2012) – http://www.mirror.co.uk/news/uk-news/\discretionary{-}{}{}ryanair-passenger-gets-on-wrong-plane-946207.
15. Bella, G., Coles-Kemp, L.: Layered analysis of security ceremonies. In: IFIP SEC, Springer (2012) 273–286

16. Bella, G., Giustolisi, R., G.Lenzini: Socio-technical formal analysis of TLS certificate validation in modern browsers. In et al., J.C.R., ed.: Proc of 11th International Conference on Privacy, Security and Trust (PST'13), IEEE Press (2013) 309–316
17. Bella, G., Coles-Kemp, L.: Internet users' security and privacy while they interact with amazon. In: Proc of IEEE International Workshop on Trust and Identity in Mobile Internet, Computing and Communications (IEEE TrustID'11), IEEE Press (2011) 878–883

Mechanisms with Verification and Fair Allocation Problems

Gianluca Greco

University of Calabria

Abstract. Whenever the outcome of some social choice process depends on the information collected from a number of self-interested agents, strategic issues come into play and mechanism design techniques have to be used in order to motivate all agents to truthfully report the relevant information they own as their private knowledge. The talk illustrates some general background on these techniques and specific methods that can be applied when some kind of verification on the declarations of the agents is possible. In particular, attention is focused on analysing a class of mechanisms that naturally arise in the context of allocation problems, by proposing to interpret them in terms of well-known solution concepts for coalitional games. Complexity issues arising in this setting are also discussed, and structural requirements are investigated which can be used to identify islands of tractability.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 8–8 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

Regular Papers

Non-Atomic One-Round Walks in Polynomial Congestion Games

Cosimo Vinci

Gran Sasso Science Institute, L'Aquila - Italy
cosimo.vinci@gssi.infn.it

Abstract. In this paper we study the approximation ratio of the solutions achieved after an ϵ-approximate one-round walk in non-atomic congestion games. Prior to this work, the solution concept of one-round walks had been studied for atomic congestion games with linear latency functions only [Christodoulou et al. 2006, Bilò et al. 2011]. We focus on polynomial latency functions, and, by exploiting the primal-dual technique [Bilò 2012], we prove that the approximation ratio is exactly $((1+\epsilon)(p+1))^{p+1}$ for every polynomial of degree p. Then, we show that, by resorting to static (resp. dynamic) resource taxation, the approximation ratio can be lowered to $(1+\epsilon)^{p+1}(p+1)^p$ (resp. $(1+\epsilon)^{p+1}(p+1)!$).

Keywords: computational social choice, algorithmic game theory, congestion games

1 Introduction

Since the end of the Twentieth Century, the computer science community has been interested in the study of complex systems populated by (numerous) selfish agents interacting with each other, and in how their selfish behavior impacts on the social welfare [24]. As examples, one may think to web users greedily sharing limited resources, or to drivers who want to move as fast as possible from a location to another along a street network.

These systems are often modeled by *congestion games* [25]. In these games, there is a set of non-cooperative selfish players sharing a set of *resources* and each resource incurs a certain *latency* to the players using it. Each player has an available set of strategies, where each strategy is a non-empty subset of resources, and aims at choosing a strategy minimizing her cost which is defined as the sum of the latencies experienced on all the selected resources. A congestion game is called *atomic* when the set of players is finite, and it is called *non-atomic* when

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 11–22 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

the set of players is infinite and the contribution of each player to the social welfare is infinitesimally small.

In congestion games, selfish behavior always leads to stable outcomes, called *Pure Nash Equilibrium* [23], in which each player cannot improve her utility by unilaterally deviating from her strategy. In general, Pure Nash Equilibria do not yield an optimal social welfare, hence, to measure the quality of these outcomes, two metrics have been successfully proposed: the *price of anarchy* [21] and the *price of stability* [2]. They are defined as the highest and the lowest ratio between the social welfare at any Nash equilibrium and the social optimum, respectively.

Besides Pure Nash Equilibria, in the setting of atomic congestion games, the notion of *one-round walks starting from the empty state* has been widely investigated due to its simplicity and effectiveness. This concept assumes that, starting from the situation in which no strategy has been specified yet, the players are processed sequentially and, at each iteration, the selected player irrevocably chooses her strategy so as to minimize her cost based on the choices of the previous ones. The *approximation ratio of one-round walks starting from the empty state*, an analogous of the price of anarchy instantiated to one-round walks, measures the quality of the outcomes produced at the end of this process [22,14].

Our Contribution. In this work, we translate the solution concept of one-round walks from atomic congestion games to non-atomic ones. We define the solution concept of ϵ-*approximate non-atomic one-round walk starting from the empty state*, in which there is a continuous flow of selfish players (instead of a discrete number of players) greedily selecting their strategies with the aim of approximately (up to a factor of $1+\epsilon$) minimizing their costs, given the choices of their predecessors. The ϵ-*approximation ratio of a non-atomic one-round walk starting from the empty state* is the highest ratio of the the social value achieved by the final outcome of an ϵ-approximate non-atomic one-round walk and the optimal social value. In particular, we study this metric for non-atomic congestion games with polynomial latencies, by proving a tight bound of $((1+\epsilon)(p+1))^{p+1}$, where p is the the degree of the polynomial functions. Given that the (exact and approximate) price of anarchy for these games has been proven to be equal to $\Theta\left(\max\left\{p/\log(p), (1+\epsilon)^{p+1}\right\}\right)$, our result shows that outcomes generated after one-round walks are tremendously worse (even asymptotically) than Pure Nash Equilibria in terms of social welfare.

For such a reason, we also focus on (resource) *taxation* [4]: an approach that has been intensively studied in the literature in order to improve the quality of the outcomes resulting from selfish behavior in congestion games. We prove that, by resorting to static taxation (taxes are constant with respect to resource congestion), the ϵ-approximation ratio drops to $(1+\epsilon)^{p+1}(p+1)^p$, thus having a good asymptotic reduction with respect to the case without taxes. By resorting to dynamic taxation (in which taxes can vary as a function of resource congestion), we lower the ϵ-approximation ratio to $(1+\epsilon)^{p+1}(p+1)! \in \Theta\left((1+\epsilon)^{p+1}(p+1)^{p+3/2}e^{-p}\right)$, thus having a further improvement.

Related Work. The inefficiency of equilibria for polynomial congestion games has been largely studied for both atomic and non-atomic congestion games. Relatively to the non-atomic setting, Roughgarden [27] proves that the price of anarchy for congestion games with polynomial latency functions of degree p is $[1 - p(p + 1)^{-(p+1)/p}]^{-1}$ and characterizes the price of anarchy for other latency functions.

Christodoulou et al. [12] and Awerbuch et al. [3] prove that the price of anarchy for linear atomic congestion games is $5/2$ for unweighted games and 2.168 for weighted games. Aland. et al [1] generalize the previous results to polynomial congestion games, and provide tight bounds asymptotically equal to $\Theta(p/\log(p))^{p+1}$ on the price of anarchy for both unweighted and weighted atomic congestion games with polynomial latency functions of degree p. Christodoulou et al. [13] provide bounds on the approximate price of anarchy and stability for both non-atomic and atomic polynomial congestion games, which are tight for the former.

Among the mechanisms used to improve the quality of outcomes in congestion games, the use of taxation has been extensively studied for several decades. The marginal cost taxation [4] has been proven to enforce an optimal solution in non-atomic congestion games with very general latency functions. In subsequent works, the existence and the computation of efficient taxes in many variants of non-atomic congestion games has been intensively studied [15,16,19,29].

Caragiannis et al. [11] study the efficiency of taxation for linear atomic congestion games, and, among the obtained results, they prove that the price of anarchy drops from to 2.168 to at least 2 for weighted congestion games. Bilò and Vinci [8] extend the previous result to polynomial congestion games, and prove that, by resorting to taxation, the ϵ-approximate price of anarchy of unweighted and weighted atomic congestion games with polynomial latency functions drops to $\mathcal{T}_{p+1}(1 + \epsilon)$, where $\mathcal{T}_{p+1}$ is the $(p + 1)$-th Touchard polynomial.

Other mechanisms used to reduce the price of anarchy in congestion games are Stackelberg Strategies [26,29,20,7,17], in which the strategies of a fraction of players can be controlled with the aim of reducing the price of anarchy.

Mirrokni and Vetta [22] initiate the study of the social welfare achieved after multiple rounds of best-responses in a Nash-dynamics for a particular class of games, and they also consider the case of a one round of best responses. Christodoulou et al. [14] study for the first time the quality of outcomes obtained after a one round of best responses in linear atomic congestion games. They prove that, if players start from an empty state, the approximation ratio is at most $2+\sqrt{5}$ for unweighted games and $4+2\sqrt{3}$ for weighted games. For the unweighted setting, Bilò et al. [6] prove that the previous upper bound is tight, improving a lower bound of 4 obtained for load balancing games by Caragiannis et al. [10]. They also consider as a social function the maximum utility among all players, and provide asymptotically matching bounds for the approximation ratio of one-round walks starting from the empty state and from an arbitrary state. Bilò [5] studies the approximation ratio for atomic congestion games with quadratic and cubic latency functions. The quality achieved by multiple rounds of best responses in linear congestion games has been studied in [14,18]. Bilò and Vinci

[8] improve via taxation the performances of ϵ-approximated one-round walks starting from the empty state in unweighted and weighted polynomial atomic congestion games.

2 Preliminaries

For any integer n, set $[n] := \{1, 2, \ldots, n\}$ and $[n]_0 := [n] \cup \{0\}$.

Non-Atomic Congestion Games. A *non-atomic congestion game* is a tuple $\mathsf{CG} = (\mathsf{N}, E, (\ell_e)_{e \in E}, (r_i)_{i \in \mathsf{N}}, (\Sigma_i)_{i \in \mathsf{N}})$, where N is a totally ordered set of $n \geq 2$ types of players, E is a set of resources, $\ell_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is the latency function of resource $e \in E$, and, given $i \in \mathsf{N}$, $r_i \in \mathbb{R}_{\geq 0}$ is the amount of players of type i and $\Sigma_i = \{S_{i,1}, \ldots, S_{i,m_i}\} \subseteq 2^E$ is the set of strategies of a player of type i, that is, each player of type i has $m_i \geq 1$ possible choices. A congestion game has polynomial latencies of degree $p \in \mathbb{N}$ when, for each $e \in E$, $\ell_e(x) := \sum_{d \in [p]_0} \alpha_{e,d} x^d$, with $\alpha_{e,d} \geq 0$ for each $d \in [p]_0$; when $p = 1$, we speak of affine latencies.

A *strategy profile* is an n-tuple $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n)$, where $\Delta_i : \Sigma_i \to \mathbb{R}_{\geq 0}$ is a function denoting, for each strategy $S_i \in \Sigma_i$, the amount $\Delta_i(S_i)$ of players of type i selecting strategy S_i, so that $\sum_{S \in \Sigma_i} \Delta_i(S) = r_i$. For a strategy profile $\boldsymbol{\Delta}$, the congestion of resource $e \in E$ in $\boldsymbol{\Delta}$, denoted as $k_e(\boldsymbol{\Delta}) := \sum_{i \in \mathsf{N}, S \in \Sigma_i : e \in S} \Delta_i(S)$, is the total amount of players using resource e in $\boldsymbol{\Delta}$. The cost of a player selecting a strategy $S_i \in \Sigma_i$ is defined as $c_{S_i}(\boldsymbol{\Delta}) = \sum_{e \in S_i} \ell_e(k_e(\boldsymbol{\Delta}))$ and each player aims at minimizing it.

Non-Atomic One-Round Walks. For any type $i \in \mathsf{N}$ of players, we extend the set Σ_i of strategies with the empty strategy $\emptyset_i$, so as to include also the cases in which some players have not chosen their strategies yet; in particular, we denote with $\boldsymbol{\emptyset}$ the empty state, that is, the strategy profile in which none of the players has chosen a strategy.

For any $\epsilon \geq 0$, a strategy $S_i^* \in \Sigma_i \setminus \{\emptyset_i\}$ is an *ϵ-approximate best-response* (ϵ-best-response, for brevity) for players of type i in $\boldsymbol{\Delta}$ if, for each $S_i' \in \Sigma_i \setminus \{\emptyset_i\}$, $c_{S_i^*}(\boldsymbol{\Delta}) \leq (1 + \epsilon) c_{S_i'}(\boldsymbol{\Delta})$. Let $M := \sum_{i \in \mathsf{N}} r_i$ and let $f : [0, M] \to \mathsf{N}$ be a right-continuous function such that $\int_{f^{-1}[\{i\}]} dx = r_i$ for each $i \in \mathsf{N}$. We call f an *ordering function*. Let $(\boldsymbol{\Delta}^{f,t})_{t \in [0,M]}$ be a family of strategy profiles such that: (1) $\sum_{S \in \Sigma_i \setminus \{\emptyset_i\}} \Delta_i^{f,t}(S) = \int_{[0,t] \cap f^{-1}[\{i\}]} dx$ for each $i \in \mathsf{N}$ and $t \in [0, M]$ (then, $\Delta_i^{f,t}(\emptyset_i) = r_i - \int_{[0,t] \cap f^{-1}[\{i\}]} dx$), (2) $\Delta_i^{f,t}(S)$ is non-decreasing in t. Such a family of strategy profiles is called a *weak one-round walk starting from the empty state*.

Observe that $\Delta_i^{f,t}(S)$ has to be necessarily a non-decreasing and Lipschitzian function with respect to t. Informally, a weak one-round walk models a family of strategy profiles generated by a flow of players sequentially selecting their strategies, in such a way, for any $t \in [0, M]$, there is an amount $\Delta_i^{f,t}(S)$ of players of type i which have already selected strategy S (Point 1), and these players cannot change their strategy (Point 2). Moreover, observe that f defines the ordering in which the players appear in the game.

A weak one-round walk is *simple* if $t \leq t' \Rightarrow f(t) \leq f(t')$, and, for each $i \in \mathsf{N}$, there exists a strategy $S_i \in \Sigma_i$ such that $\Delta_i^{f,M}(S_i) = \int_{f^{-1}[\{i\}]} dx$. Informally, a weak one-round walk is simple if all players select their strategy according to the ordering by which their types are defined in N, and if players of the same type play the same strategy. A simple weak one-round walk can be univocally represented as a sequence of strategies $(S_i)_{i \in \mathsf{N}}$ such that S_i is the strategy played by a player of type i. The strategy profile generated by a weak one-round walk is $\boldsymbol{\Delta}^{f,M}$.

A weak one-round walk $(\boldsymbol{\Delta}^{f,t})_{t \in [0,M]}$ is an *ϵ-approximate non-atomic one-round walk starting from the empty state* (ϵ-one-round walk, for brevity) if, for any $t \in [0, M]$, $\Delta_{f(t)}^{f,t}(S)$ is right-increasing at t only if strategy S is an ϵ-best-response in $\boldsymbol{\Delta}^{f,t}$ for player $f(t)$. Informally, an ϵ-one-round walk is a weak one-round walk in which all players sequentially select an ϵ-best-response.

Efficiency Metrics. A social function that is usually used as a measure of the quality of a strategy profile in non-atomic congestion games, is the *total latency*, defined as $\mathsf{TL}(\boldsymbol{\Delta}) = \sum_{i \in \mathsf{N}, S_i \in \Sigma} \Delta_i(S_i) \cdot c_{S_i}(\boldsymbol{\Delta}) = \sum_{e \in E} k_e(\boldsymbol{\Delta}) \cdot \ell_e(k_e(\boldsymbol{\Delta}))$. A *social optimum* is a strategy profile $\boldsymbol{\Delta}^*$ minimizing TL. The *approximation ratio of ϵ-approximate non-atomic one-round walks starting from the empty state* (ϵ-approximation ratio, for brevity) of a congestion game CG (resp. a class of congestion games) with respect to the total latency is the supremum of the ratio $\mathsf{TL}(\boldsymbol{\Delta})/\mathsf{TL}(\boldsymbol{\Delta}^*)$, where $\boldsymbol{\Delta}$ is the strategy profile generated by an ϵ-one-round walk for CG (resp. for some game CG in the class) and $\boldsymbol{\Delta}^*$ is a social optimum for CG.

Taxes. A *dynamic tax-function* $T := (T_e)_{e \in E}$ is a class of functions $T_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ increasing the latency function perceived by each player. The presence of T determines a new congestion game $\mathsf{CG}(T) = (\mathsf{N}, E, (\ell'_e)_{e \in E}, (r_i)_{i \in \mathsf{N}}, (\Sigma_i)_{i \in \mathsf{N}})$, equal to CG except the latency functions, such that $\ell'_e(x) := \ell_e(x) + T_e(x)$ for each $e \in E$. However, the total latency function is still evaluated with respect to the initial latency functions ℓ_es. T is a *static tax-function* if each T_e is a constant function (with respect to the resource congestion).

3 Approximation Ratio of One-Round Walks

3.1 Upper Bound

We prove our upper bound on the ϵ-approximation ratio by using the primal-dual method: a technique introduced by Bilò in [5] to prove bounds on the performance guarantee of self-emerging solutions (such as approximate pure Nash equilibria and their generalizations, approximate one-round walks, and so on) in atomic and non-atomic congestion games.

In our setting, we want to establish an upper bound on the worst-case performance guarantee of ϵ-one-round walks with respect to the total latency in the class of polynomial congestion games. For a general, but fixed, congestion game

CG, the method requires the construction of a linear program $\mathsf{LP}(\mathsf{CG})$ based on the following steps: (1) the objective function is $\mathsf{TL}(\boldsymbol{\Delta})$, where $\boldsymbol{\Delta}$ is the strategy profile generated by an arbitrary fixed ϵ-one-round walk for CG, (2) add the constraint $\mathsf{TL}(\boldsymbol{\Delta}^*) = 1$, where $\boldsymbol{\Delta}^*$ is an arbitrary fixed social optimal for CG, (3) relate $\boldsymbol{\Delta}$ and $\boldsymbol{\Delta}^*$ by adding, for each $i \in \mathsf{N}$, suitable constraints that need to be satisfied by the choices performed by all players of type i, (4) the coefficients of the latency functions are treated as variables, while all the other quantities (e.g. resources congestions) are treated as fixed parameters.

By the generality of CG, it follows that the optimal solution of $\mathsf{LP}(\mathsf{CG})$ is an upper bound on the ϵ-approximation ratio in polynomial congestion games. By the weak-duality theorem [9], any feasible solution to the dual formulation of $\mathsf{LP}(\mathsf{CG})$ yields an upper bound on the ϵ-approximation ratio in polynomial congestion games. The more the constraints defined during step (3) provide an accurate characterization of the properties of ϵ-one-round walks, the more the achieved upper bound will be significant (and possibly tight).

Theorem 1. *The approximation ratio of ϵ-approximate one-round walks starting from the empty state in congestion games with polynomial latency functions of degree p is at most $((1+\epsilon)(p+1))^{p+1}$.*

Proof. For an integer $p \geq 1$, fix a congestion game CG having polynomial latencies of degree p. Let f be an ordering function such that $(\boldsymbol{\Delta}^{f,t})_{t\in[0,M]}$ is an ϵ-approximate one-round walk starting from the empty state, and let $\boldsymbol{\Delta} := \boldsymbol{\Delta}^{f,M}$. Let $\boldsymbol{\Delta}^*$ be an optimal strategy profile, and let $(o(t))_{t\in[0,M]}$ be a family of strategies such that $o(t)$ is right-continuous with respect to t, $o(t)$ is a strategy of player $f(t)$ and $\int_{o^{-1}[\{S\}]\cap f^{-1}[\{i\}]} dt = \Delta_i^*(S)$. Observe that such a family of strategies always exists. For the sake of conciseness, we set $k_e := k_e(\boldsymbol{\Delta})$ and $o_e := k_e(\boldsymbol{\Delta}^*)$. By applying the primal-dual method, we get the following linear program $\mathsf{LP}(\mathsf{CG})$:

$$\max \quad \underbrace{\sum_{e\in E}\sum_{d=0}^{p}\alpha_{e,d}k_e^{d+1}}_{\mathsf{TL}(\boldsymbol{\Delta})} \tag{1}$$

$$s.t. \quad c^{\epsilon}(S^*, o(t), \boldsymbol{\Delta}^{f,t}) \leq 0, \quad \forall S^* \in \Sigma^{\epsilon}_{f(t)}(\boldsymbol{\Delta}^{f,t}), \quad \forall t \in [0,M] \tag{2}$$

$$\underbrace{\sum_{e\in E}\sum_{d=0}^{p}\alpha_{e,d}o_e^{d+1}}_{\mathsf{TL}(\boldsymbol{\Delta}^*)} = 1 \tag{3}$$

$$\alpha_{e,d} \geq 0, \quad \forall e \in E, \forall d \in [p]_0 \tag{4}$$

where $\Sigma_i^{\epsilon}(\boldsymbol{\Delta}')$ is the set of all ϵ-best-responses of players of type i in a strategy profile $\boldsymbol{\Delta}'$, and $c^{\epsilon}(S, S', \boldsymbol{\Delta}')$ is defined as follows:

$$c^{\epsilon}(S, S', \boldsymbol{\Delta}') := \underbrace{\sum_{e\in S}\sum_{d=0}^{p}\alpha_{e,d}k_e^d(\boldsymbol{\Delta}')}_{c_S(\boldsymbol{\Delta}')} - (1+\epsilon)\underbrace{\sum_{e\in S'}\sum_{d=0}^{p}\alpha_{e,d}k_e^d(\boldsymbol{\Delta}')}_{c_{S'}(\boldsymbol{\Delta}')} \tag{5}$$

The constraints (2) come from the definition of ϵ-best-responses and ϵ-one-round walks. By replacing constraint (2) with

$$\hat{c}^{\epsilon}(S^*, o(t), \boldsymbol{\Delta}^{f,t}) := \sum_{e \in S^*} \sum_{d=0}^{p} \alpha_{e,d} k_e^d(\boldsymbol{\Delta}^{f,t}) - (1+\epsilon) \sum_{e \in o(t)} \sum_{d=0}^{p} \alpha_{e,d} k_e^d(\boldsymbol{\Delta}) \leq 0 \quad (6)$$

we obtain a relaxation of $\mathsf{LP}(\mathsf{CG})$. Given $i \in \mathsf{N}$ and $S \in \Sigma_i \setminus \emptyset_i$, $\hat{c}^{\epsilon}(S, o(t), \boldsymbol{\Delta}^{f,t})$ can be defined as a function of $\Delta_i^{f,t}(S)$ except for the intervals on which $\Delta_i^{f,t}(S)$ is constant. By exploiting (6), we obtain

$$0 \geq \sum_{i \in \mathsf{N}, S \in \Sigma_i} \int_0^{\Delta_i(S)} \hat{c}^{\epsilon}(S, o(t), \boldsymbol{\Delta}^{f,t}) d(\Delta_i^{f,t}(S)) \quad (7)$$

$$= \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} \int_0^{k_e} k_e^d(\boldsymbol{\Delta}^{f,t}) d \left(\sum_{i \in \mathsf{N}, S \in \Sigma_i : e \in S} \left(\Delta_i^{f,t}(S) \right) \right) \quad (8)$$

$$- (1+\epsilon) \int_0^{M} \left(\sum_{e \in o(t)} \sum_{d=0}^{p} \alpha_{e,d} k_e^d \right) dt \quad (9)$$

$$= \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} \int_0^{k_e} k_e^d(\boldsymbol{\Delta}^{f,t}) d(k_e(\boldsymbol{\Delta}^{f,t})) - (1+\epsilon) \sum_{e \in E} \int_{\{t : e \in o(t)\}} \left(\sum_{d=0}^{p} \alpha_{e,d} k_e^d \right) dt \quad (10)$$

$$= \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} \int_0^{k_e} k^d dk - (1+\epsilon) \sum_{e \in E} \int_0^{o_e} \left(\sum_{d=0}^{p} \alpha_{e,d} k_e^d \right) dk \quad (11)$$

$$= \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} \frac{{k_e}^{d+1}}{d+1} - (1+\epsilon) \sum_{e \in E} o_e \sum_{d=0}^{p} \alpha_{e,d} k_e^d \quad (12)$$

The equivalences between all pairs of integrals in the above derivation (along with the fact that they are well-defined) come from the properties of the $\Delta_i^{f,t}(S)$s as functions of t, and, more generally, from the Lebesgue theory of integration [28]. In conclusion, by replacing the left-hand side of (2) with (12), we obtain the following relaxation of $\mathsf{LP}(\mathsf{CG})$:

$$\max \quad \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} k_e^{d+1} \quad (13)$$

$$s.t. \quad \sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} \frac{{k_e}^{d+1}}{d+1} - (1+\epsilon) \sum_{e \in E} o_e \sum_{d=0}^{p} \alpha_{e,d} k_e^d \leq 0 \quad (14)$$

$$\sum_{e \in E} \sum_{d=0}^{p} \alpha_{e,d} o_e^{d+1} = 1 \quad (15)$$

$$\alpha_{e,d} \geq 0, \quad \forall e \in E, \forall d \in [p]_0 \quad (16)$$

The dual of this problem is the following linear problem $\mathsf{LP}'(\mathsf{CG})$:

$$\min \quad \gamma \tag{17}$$

$$s.t. \quad x\left(\frac{k_e^{d+1}}{d+1} - (1+\epsilon)o_e k_e^d\right) + \gamma o_e^{d+1} \geq k_e^{d+1} \quad \forall e \in E, d \in [p]_0 \tag{18}$$

$$x, \gamma \geq 0 \tag{19}$$

Clearly, any feasible solution for $\mathsf{LP}'(\mathsf{CG})$ is an upper bound on the ϵ-approximation ratio. We show that, by setting $\gamma := ((1+\epsilon)(p+1))^{p+1}$ and $x := (p+1)^2$, the constraints (18) are always satisfied. Indeed, if $o_e = 0$, the constraints (18) are satisfied. Conversely, if $o_e > 0$, by rewriting the constraints (18) in terms of the quantity $t_e := k_e/o_e$, we get the following inequality:

$$g(t_e, d) := -t_e^{d+1} + (p+1)^2\left(\frac{t_e^{d+1}}{d+1} - (1+\epsilon)t_e^d\right) + ((1+\epsilon)(p+1))^{p+1} \geq 0 \tag{20}$$

for each $e \in E$, $d \in [p]_0$ and $t_e \geq 0$. This inequality is always satisfied for each $d \in [p]_0$ and $t_e \geq 0$ (see the full version), and this fact completes the proof of the theorem. □

3.2 Lower Bound

Theorem 2. *For each $\delta > 0$ there exists a congestion game having polynomial latency functions of degree p such that the approximation ratio of ϵ-approximate one-round walks starting from the empty state is higher than $((1+\epsilon)(p+1))^{p+1} - \delta$.*

Proof. Let $n, m \in \mathbb{N}$ and $q := \lfloor (1+\epsilon)(p+1)n \rfloor - 1$. Let $\mathsf{CG}_{m,n}$ be a congestion game such that the set of players' types is $\mathsf{N} := \{a_1^* \dots, a_{q+1}^*, a_1', a_1, \dots, a_m', a_m\}$, the set of resources is $E := \{e_1, \dots, e_{n+m+q-1}\}$, the amount of players of type a_1^* and types a_i's is 2, while the amount of players of the remaining types is 1. Players of type a_i^* can only select the strategy $s_i^* := \{e_1, e_2, \dots, e_{n+i-2}\}$, players of type a_i' can only select the strategy $s_i' := \{e_{n+i-1}\}$, and players of type a_i can select $s_i := \{e_{n+i}, e_{n+i+1}, \dots, e_{n+i+q-1}\}$ or $o_i := \{e_i, e_{i+1}, \dots, e_{n+i-1}\}$. The latency function is $\ell_e(k_e) := k_e^p/n^{p+1}$ for each resource $e \in E$.

Consider the simple weak one-round walk

$$(\boldsymbol{\Delta}^{f,t})_{t\in[0,M]} := (s_1^* \dots, s_{q+1}^*, s_1', s_1, \dots, s_m', s_m).$$

To prove that $(\boldsymbol{\Delta}^{f,t})_{t\in[0,M]}$ is an ϵ-one-round walk it is sufficient to show that s_i is an ϵ-best-response in $\boldsymbol{\Delta}^{f,t}$ for each t such that $f(t) = a_i$. Given such a t, we get

$$c_{s_i}(\boldsymbol{\Delta}^{f,t}) \leq \sum_{j=1}^{q} \frac{j^p}{n^{p+1}} \leq \int_0^{(1+\epsilon)(p+1)n} \frac{1}{n}\left(\frac{x}{n}\right)^p dx = \int_0^{(1+\epsilon)(p+1)} y^p dy \tag{21}$$

$$= (1+\epsilon)^{p+1}(p+1)^p \leq (1+\epsilon)\left(\frac{q+2}{n}\right)^p \tag{22}$$

$$= (1+\epsilon)n\frac{1}{n}\left(\frac{q+2}{n}\right)^p = (1+\epsilon)c_{o_i}(\boldsymbol{\Delta}^{f,t}) \tag{23}$$

We conclude that s_i is an ϵ-best-response in the strategy profile $\boldsymbol{\Delta}^{f,t}$ for a player of type $a_i = f_1(t)$. Therefore, $(\boldsymbol{\Delta}^{f,t})_{t\in[0,M]}$ is an ϵ-one-round walk, and generates a strategy profile $\boldsymbol{\Delta}_{m,n}$ in which all players of type a_i select the strategy s_i. Let ${\boldsymbol{\Delta}^*}_{m,n}$ be the strategy profile in which each player of type a_i selects the strategy o_i.

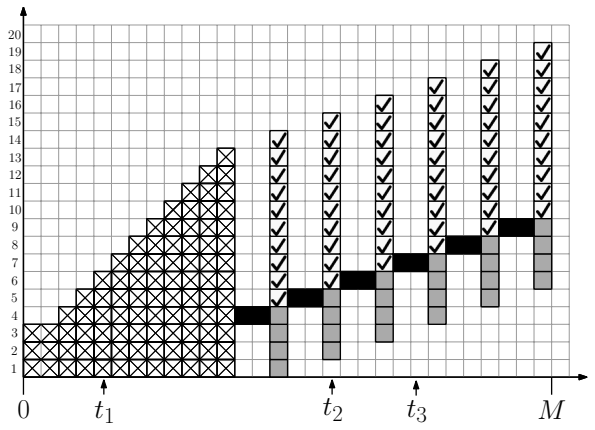


Fig. 1. The figure depicts a lower bounding instance with $n = 4$, $m = 6$, $\ell(k_e) = k_e/4$ and $\epsilon = 0.4$. Observe that $q = \lfloor 1.4 \cdot 2 \cdot 4 \rfloor - 1 = 10$ and the total amount of players is $M = 3m + 2 + q = 30$. The abscissa represents the amount of players which already selected their strategy, and the ordinate represents the resources. The ordinate values of coloured squares having abscissa $x \in [30]$ denote the strategies of all players of type $f(t)$ with $t \in [x-1, x)$. The squares labeled with a cross/a black square/a dark grey square/a tic are related to the resources selected in the the strategies $s_i^*/s_i'/o_i/s_i$. For instance, when an amount $t_1/t_2/t_3$ of players have already entered in the game, a player of type $a_4^*/a_2/a_4'$ can select the strategy $s_4^* = \{e_1, e_2, \ldots, e_6\}$ (crosses)/$s_2 = \{e_6, e_7, \ldots, e_{15}\}$ (tics) or $o_2 = \{e_2, e_3, \ldots, e_5\}$ (dark grey squares)/$s_4' = \{e_7\}$ (black squares).

We get ($o(m)$ is related to the usual asymptotic notation):

$$\lim_{n\to\infty}\lim_{m\to\infty}\frac{\mathsf{TL}(\boldsymbol{\Delta}_{m,n})}{\mathsf{TL}({\boldsymbol{\Delta}^*}_{m,n})} \geq \lim_{n\to\infty}\lim_{m\to\infty}\frac{q\frac{1}{n}\left(\frac{q}{n}\right)^p(m-o(m))+o(m)}{\left(\frac{n+2}{n}\right)^{p+1}(m-o(m))+o(m)} \tag{24}$$

$$= \lim_{n\to\infty}\left(\frac{q}{n}\right)^{p+1} = \lim_{n\to\infty}\left(\frac{\lfloor(1+\epsilon)(p+1)n\rfloor-1}{n}\right)^{p+1} \tag{25}$$

$$= ((1+\epsilon)(p+1))^{p+1}. \tag{26}$$

which completes the proof. □

4 Efficiency of Taxation

In the following theorems, we prove that by resorting to static or dynamic taxation the ϵ-approximation ratio can be consistently lowered.

Theorem 3. *The approximation ratio of ϵ-approximate one-round walks starting from the empty state in congestion games with polynomial latency functions of degree p is at most $(1+\epsilon)^{p+1}(p+1)^p$ by resorting to the static tax-function:*

$$T_e := \alpha_{e,p} o_e^p \xi, \quad \textit{with} \quad \xi := \frac{(1+\epsilon)^p (p+1)^{p-1}}{p}. \tag{27}$$

Proof. For an integer $p \geq 1$, fix a congestion game $\mathsf{CG}(T)$, where T is the tax-function defined in (27). Define $(\boldsymbol{\Delta}^{f,t})_{t\in[0,M]}$, $o(t)$, $\boldsymbol{\Delta}$, $\boldsymbol{\Delta}^*$, k_e and o_e as in the proof of Theorem 1. As done in the proof of Theorem 1, by applying the primal-dual method and by relaxing the constraints modelling ϵ-best-responses (which take into account taxation in this case), we get the following linear program:

$$\max \quad \sum_{e\in E}\sum_{d=0}^{p} \alpha_{e,d} k_e^{d+1} \tag{28}$$

$$s.t. \quad \sum_{e\in E}\left(\sum_{d=0}^{p}\alpha_{e,d}\left(\frac{{k_e}^{d+1}}{d+1} - (1+\epsilon)o_e k_e^d\right) + \alpha_{e,p}\left(o_e^p \xi k_e - o_e^{p+1}\xi\right)\right) \leq 0 \tag{29}$$

$$\sum_{e\in E}\sum_{d=0}^{p}\alpha_{e,d} o_e^{d+1} = 1 \tag{30}$$

$$\alpha_{e,d} \geq 0, \quad \forall e \in E, \forall d \in [p]_0 \tag{31}$$

The dual program is

$$\min \quad \gamma \tag{32}$$

$$s.t. \quad x\left(\frac{k_e^{d+1}}{d+1} - (1+\epsilon)o_e k_e^d\right) + \gamma o_e^{d+1} \geq k_e^{d+1} \quad \forall e \in E, d \in [p-1]_0 \tag{33}$$

$$x\left(\frac{k_e^{p+1}}{p+1} + \xi o_e^p k_e - (1+\epsilon)o_e\left(k_e^p + \xi o_e^p\right)\right) + \gamma o_e^{p+1} \geq k_e^{p+1} \quad \forall e \in E \tag{34}$$

$$x, \gamma \geq 0 \tag{35}$$

By setting $\gamma := (1+\epsilon)^{p+1}(p+1)^p$ and $x := (p+1)p$, the constraints (33) and (34) are satisfied (see the full version), and the claim follows. □

Theorem 4. *The approximation ratio of ϵ-approximate one-round walks starting from the empty state in congestion games with polynomial latency functions of degree p is at most $(1+\epsilon)^{p+1}(p+1)!$ by resorting to the dynamic tax-function (set $(d)_j := d!/(d-j)!$):*

$$T_e(k_e) := \sum_{d=0}^{p}\alpha_{e,d} T_{e,d}(k_e) \quad \textit{with} \quad T_{e,d}(k_e) := \sum_{j=1}^{d}(1+\epsilon)^j (d)_j k_e^{d-j} o_e^j \tag{36}$$

Proof. By reconsidering the proof of Theorem 3, but with the tax (36) instead of the tax (27), we get a dual program having the following constraints:

$$-k_e^{d+1} + x\left(\frac{k_e^{d+1}}{d+1} + \int_0^{k_e} T_{e,d}(u)du - (1+\epsilon)o_e(k_e^d + T_{e,d}(k))\right) + \gamma o_e^{d+1} \geq 0 \tag{37}$$

for each $e \in E$ and $d \in [p]_0$. These constraints are always satisfied if $x := p+1$ and $\gamma := (1+\epsilon)^{p+1}(p+1)!$ (see the full version), and the claim follows. □

Remark 1. Observe that finding an optimal strategy profile requires to solve a convex minimization problem. Therefore, for each $\delta > 0$, one can compute in polynomial time a strategy profile whose social value is at most $1+\delta$ times the social optimum [9]. Then, if $\tilde{o}_e$ is the congestion of resource e in that strategy profile, by using the tax (27) (resp. (36)) with $\tilde{o}_e$ in place of o_e, one can easily prove that the resulting ϵ-approximation ratio is at most $(1+\delta)$ times that of Theorem 3 (resp. Theorem 4).

References

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact price of anarchy for polynomial congestion games. SIAM J. Comput. 40(5), 1211–1233 (2011)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. pp. 295–304. FOCS, IEEE Computer Society (2004)
3. Awerbuch, B., Azar, Y., Epstein, A.: The Price of Routing Unsplittable Flow. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing. pp. 57–66. STOC, ACM (2005)
4. Beckmann, M.: Studies in the economics of transportation. Yale University Press for the Cowles Commission for Research in Economics New Haven (1959)
5. Bilò, V.: A Unifying tool for Bounding the quality of Non-Cooperative Solutions in Weighted Congestion Games. Approximation and Online Algorithms: 10th International Workshop, WAOA 2012 pp. 215–228 (2013)
6. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: Performance of one-round walks in linear congestion games. Theor. Comp. Sys. 49(1), 24–45 (2011)
7. Bilò, V., Vinci, C.: On stackelberg strategies in affine congestion games. In: Web and Internet Economics: 11th International Conference, WINE 2015. pp. 132–145. Springer Berlin Heidelberg (2015)
8. Bilò, V., Vinci, C.: Dynamic taxes for polynomial congestion games. In: Proceedings of the 17th ACM Conference on Electronic Commerce. EC (2016), To appear.
9. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York, NY, USA (2004)
10. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight bounds for selfish and greedy load balancing. Algorithmica 61(3), 606–637 (2011)
11. Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: Taxes for Linear Atomic Congestion Games. ACM Trans. Algorithms 7(1), 13:1–13:31 (2010)

12. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing. pp. 67–73. STOC '05, ACM (2005)
13. Christodoulou, G., Koutsoupias, E., Spirakis, P.G.: On the performance of approximate equilibria in congestion games. Algorithmica 61(1), 116–140 (2011)
14. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and approximation in potential games. In: Proceedings of the 23th Annual Conference on Theoretical Aspects of Computer Science. pp. 349–360. STACS, Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
15. Cole, R., Dodis, Y., Roughgarden, T.: How much can taxes help selfish routing? In: Proceedings of the 4th ACM Conference on Electronic Commerce. pp. 98–107. EC, ACM (2003)
16. Cole, R., Dodis, Y., Roughgarden, T.: Pricing Network Edges for Heterogeneous Selfish Users. In: Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing. pp. 521–530. STOC, ACM (2003)
17. Fanelli, A., Flammini, M., Moscardelli, L.: Stackelberg Strategies for Network Design Games. In: Proceedings of the 6th International Conference on Internet and Network Economics. pp. 222–233. WINE, Springer-Verlag (2010)
18. Fanelli, A., Flammini, M., Moscardelli, L.: The speed of convergence in congestion games under best-response dynamics. ACM Trans. Algorithms 8(3), 25:1–25:15 (2012)
19. Fleischer, L., Jain, K., Mahdian, M.: Tolls for Heterogeneous Selfish Users in Multicommodity Networks and Generalized Congestion Games. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. pp. 277–285. FOCS, IEEE Computer Society (2004)
20. Fotakis, D.: Stackelberg Strategies for Atomic Congestion Games. Theor. Comp. Sys. 47(1), 218–249 (2010)
21. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science. pp. 404–413. STACS, Springer-Verlag (1999)
22. Mirrokni, V.S., Vetta, A.: Convergence issues in competitive games. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 183–194. Springer Berlin Heidelberg (2004)
23. Nash, J.F.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences of the United States of America 36(1), 48–49 (1950)
24. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.): Algorithmic Game Theory (2007)
25. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
26. Roughgarden, T.: Stackelberg scheduling strategies. In: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing. pp. 104–113. STOC '01, ACM (2001)
27. Roughgarden, T.: The price of anarchy is independent of the network topology. J. Comput. Syst. Sci. 67(2), 341–364 (2003)
28. Rudin, W.: Real and Complex Analysis, 3rd Ed. McGraw-Hill, Inc., New York, NY, USA (1987)
29. Swamy, C.: The Effectiveness of Stackelberg Strategies and Tolls for Network Congestion Games. ACM Trans. Algorithms 8(4), 36:1–36:19 (2012)

Conjunctive Query Answering via a Fragment of Set Theory*

Domenico Cantone, Marianna Nicolosi-Asmundo, and
Daniele Francesco Santamaria

University of Catania, Dept. of Mathematics and Computer Science
email: {cantone,nicolosi,santamaria}@dmi.unict.it

Abstract. We address the problem of Conjunctive Query Answering (CQA) for the description logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R},\times}\rangle(\mathbf{D})$ ($\mathcal{DL}^{4,\times}_{\mathbf{D}}$, for short) which extends the logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R}}\rangle(\mathbf{D})$ with Boolean operations on concrete roles and with the product of concepts.
The result is obtained by formalizing $\mathcal{DL}^{4,\times}_{\mathbf{D}}$-knowledge bases and $\mathcal{DL}^{4,\times}_{\mathbf{D}}$-conjunctive queries in terms of formulae of the four-level set-theoretic fragment $\mathsf{4LQS}^{\mathsf{R}}$, which admits a restricted form of quantification on variables of the first three levels and on pair terms. We solve the CQA problem for $\mathcal{DL}^{4,\times}_{\mathbf{D}}$ through a decision procedure for the satisfiability problem of $\mathsf{4LQS}^{\mathsf{R}}$. We further define a KE-tableau based procedure for the same problem, more suitable for implementation purposes, and analyze its computational complexity.

1 Introduction

In the last few years, results from Computable Set Theory have been used as a means to represent and reason about description logics and rule languages for the semantic web. For instance, in [4–6], fragments of set theory with constructs related to *multi-valued maps* have been studied and applied to the realm of knowledge representation. In [8], an expressive description logic, called $\mathcal{DL}\langle\mathsf{MLSS}^{\times}_{2,m}\rangle$, has been introduced and the consistency problem for $\mathcal{DL}\langle\mathsf{MLSS}^{\times}_{2,m}\rangle$-knowledge bases has been proved **NP**-complete. The description logic $\mathcal{DL}\langle\mathsf{MLSS}^{\times}_{2,m}\rangle$ has been extended with additional constructs and SWRL rules in [6], proving that the decision problem for the resulting logic, called $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0},\mathbf{2}}\rangle$, is still **NP**-complete under suitable conditions. The description logic $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0},\mathbf{2}}\rangle$ has been extended with some *metamodelling* features in [4]. In [7], the description logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R}}\rangle(\mathbf{D})$ (more simply referred to as $\mathcal{DL}^{4}_{\mathbf{D}}$) has been introduced. $\mathcal{DL}^{4}_{\mathbf{D}}$ can be represented

* This work has been partially supported by the Polish National Science Centre research project DEC-2011/02/A/HS1/00395.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 23–35 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

in the decidable four-level stratified fragment of set theory $\mathsf{4LQS^R}$ involving a restricted form of quantification over variables of the first three levels and pair terms (cf. [2]). The logic $\mathcal{DL}^4_{\mathbf{D}}$ admits concept constructs such as full negation, union and intersection of concepts, concept domain and range, existential quantification and min cardinality on the left-hand side of inclusion axioms. It also supports role constructs such as role chains on the left hand side of inclusion axioms, union, intersection, and complement of abstract roles, and properties on roles such as transitivity, symmetry, reflexivity, and irreflexivity. It admits datatypes, a simple form of concrete domains that are relevant in real world applications.

The consistency problem for $\mathcal{DL}^4_{\mathbf{D}}$-knowledge bases has been proved decidable in [7] by means of a reduction to the satisfiability problem for $\mathsf{4LQS^R}$, proved decidable in [2]. It has also been proved, under not very restrictive constraints, that the consistency problem for $\mathcal{DL}^4_{\mathbf{D}}$-knowledge bases is **NP**-complete. The latter result has practical outcomes since, for example, the ontology Ontoceramic [9] can be expressed in such a restricted version of $\mathcal{DL}^4_{\mathbf{D}}$. Finally, we mention that the papers [4–8] are concerned with traditional research issues for description logics mainly focused on the parts of a knowledge base representing conceptual information, namely the TBox and the RBox, where the principal reasoning services are subsumption and satisfiability.

In this paper we exploit decidability results presented in [2, 7] to deal with reasoning services for knowledge bases involving ABoxes. The most basic service to query the instance data is *instance retrieval*, i.e., the task of retrieving all individuals that instantiate a class C, and, dually, all named classes C that an individual belongs to. In particular, a powerful way to query ABoxes is the *Conjunctive Query Answering* task (CQA). CQA is relevant in the context of description logics and, in particular, for real world applications based on semantic web technologies, since it provides a mechanism allowing users and applications to interact with ontologies and data. The task of CQA has been studied for several well-known description logics (cf. [1, 13, 15]).

In particular, we introduce the description logic $\mathcal{DL}\langle\mathsf{4LQS^{R,\times}}\rangle(\mathbf{D})$ ($\mathcal{DL}^{4,\times}_{\mathbf{D}}$, for short), extending $\mathcal{DL}^4_{\mathbf{D}}$ with Boolean operations on concrete roles and with the product of concepts. Then we define the CQA problem for $\mathcal{DL}^{4,\times}_{\mathbf{D}}$ and prove its decidability via a reduction to the CQA problem for $\mathsf{4LQS^R}$, whose decidability follows from that of the satisfiability problem for $\mathsf{4LQS^R}$ (proved in [2]). Finally, we present a KE-tableau based procedure that, given a $\mathcal{DL}^{4,\times}_{\mathbf{D}}$-query Q and a $\mathcal{DL}^{4,\times}_{\mathbf{D}}$-knowledge base $\mathcal{KB}$ represented in set-theoretic terms, determines the answer set of Q with respect to $\mathcal{KB}$, providing also some complexity results. The choice of the KE-tableau system [10] is motivated by the fact that this variant of the tableau method allows one to construct trees whose distinct branches define mutually exclusive situations thus avoiding the proliferation of redundant branches, typical of semantic tableaux.

2 Preliminaries

2.1 The set-theoretic fragment 4LQS$^{\mathsf{R}}$

It is convenient to first introduce the syntax and semantics of a more general four-level quantified language, denoted 4LQS. Then we provide some restrictions on quantified formulae of 4LQS that characterize 4LQS$^{\mathsf{R}}$. We recall that the satisfiability problem for 4LQS$^{\mathsf{R}}$ has been proved decidable in [2].

4LQS involves four collections, $\mathcal{V}_i$, of variables of sort i, for $i = 0, 1, 2, 3$. Variables of sort i, for $i = 0, 1, 2, 3$, will be denoted by $X^i, Y^i, Z^i, \ldots$ (in particular, variables of sort 0 will also be denoted by $x, y, z, \ldots$). In addition to variables, 4LQS involves also *pair terms* of the form $\langle x, y\rangle$, with $x, y \in \mathcal{V}_0$.

4LQS-*quantifier-free atomic formulae* are classified as:

- level 0: $x = y$, $x \in X^1$, $\langle x, y\rangle = X^2$, $\langle x, y\rangle \in X^3$;
- level 1: $X^1 = Y^1$, $X^1 \in X^2$;
- level 2: $X^2 = Y^2$, $X^2 \in X^3$.

4LQS *purely universal formulae* are classified as:

- level 1: $(\forall z_1)\ldots(\forall z_n)\varphi_0$, where $z_1, \ldots, z_n \in \mathcal{V}_0$ and φ_0 is any propositional combination of quantifier-free atomic formulae of level 0;
- level 2: $(\forall Z_1^1)\ldots(\forall Z_m^1)\varphi_1$, where $Z_1^1, \ldots, Z_m^1 \in \mathcal{V}_1$ and φ_1 is any propositional combination of quantifier-free atomic formulae of levels 0 and 1, and of purely universal formulae of level 1;
- level 3: $(\forall Z_1^2)\ldots(\forall Z_p^2)\varphi_2$, where $Z_1^2, \ldots, Z_p^2 \in \mathcal{V}_2$ and φ_2 is any propositional combination of quantifier-free atomic formulae and of purely universal formulae of levels 1 and 2.

4LQS-formulae are all the propositional combinations of quantifier-free atomic formulae of levels 0, 1, 2 and of purely universal formulae of levels 1, 2, 3.

Let φ be a 4LQS-formula. Without loss of generality, we can assume that φ contains only $\neg$, $\wedge$, $\vee$ as propositional connectives. Further, let S_φ be the syntax tree for a 4LQS-formula φ,[1] and let ν be a node of S_φ. We say that a 4LQS-formula ψ occurs within φ at position ν if the subtree of S_φ rooted at ν is identical to S_ψ. In this case we refer to ν as an occurrence of ψ in φ and to the path from the root of S_φ to ν as its occurrence path. An occurrence of ψ within φ is *positive* if its occurrence path deprived by its last node contains an even number of nodes labelled by a 4LQS-formula of type $\neg\chi$. Otherwise, the occurrence is said to be *negative*.

The variables $z_1, \ldots, z_n$ are said to occur *quantified* in $(\forall z_1)\ldots(\forall z_n)\varphi_0$. Likewise, $Z_1^1, \ldots, Z_m^1$ and $Z_1^2, \ldots, Z_p^2$ occur quantified in $(\forall Z_1^1)\ldots(\forall Z_m^1)\varphi_1$ and in $(\forall Z_1^2)\ldots(\forall Z_p^2)\varphi_2$, respectively. A variable occurs *free* in a 4LQS-formula φ if it does not occur quantified in any subformula of φ. For $i = 0, 1, 2, 3$, we denote with $\mathtt{Var}_i(\varphi)$ the collections of variables of level i occurring free in φ.

[1] The notion of syntax tree for 4LQS-formulae is similar to the notion of syntax tree for formulae of first-order logic. A precise definition of the latter can be found in [11].

A (level 0) substitution $\sigma \coloneqq \{x_1/y_1, \ldots, x_n/y_n\}$ is the mapping $\varphi \mapsto \varphi\sigma$ such that, for any given 4LQS-formula φ, $\varphi\sigma$ is the 4LQS-formula obtained from φ by replacing the free occurrences of the variables $x_1, \ldots, x_n$ in φ with the variables $y_1, \ldots, y_n$, respectively. We say that a substitution σ is free for φ if the formulae φ and $\varphi\sigma$ have exactly the same occurrences of quantified variables.

A 4LQS-*interpretation* is a pair $\boldsymbol{\mathcal{M}} = (D, M)$ where D is a non-empty collection of objects (called *domain* or *universe* of $\boldsymbol{\mathcal{M}}$) and M is an assignment over the variables in $\mathcal{V}_i$, for $i = 0, 1, 2, 3$, such that: $MX^0 \in D, MX^1 \in \mathcal{P}(D), MX^2 \in \mathcal{P}(\mathcal{P}(D)), MX^3 \in \mathcal{P}(\mathcal{P}(\mathcal{P}(D)))$, where $X^i \in \mathcal{V}_i$, for $i = 0, 1, 2, 3$, and $\mathcal{P}(s)$ denotes the powerset of s.

Pair terms are interpreted *à la* Kuratowski, and therefore we put

$$M\langle x, y\rangle \coloneqq \{\{Mx\}, \{Mx, My\}\}.$$

Next, let

- $\boldsymbol{\mathcal{M}} = (D, M)$ be a 4LQS-interpretation,
- $x_1, \ldots, x_n \in \mathcal{V}_0, \quad X_1^1, \ldots, X_m^1 \in \mathcal{V}_1, \quad X_1^2, \ldots, X_p^2 \in \mathcal{V}_2$, and
- $u_1, \ldots, u_n \in D, \quad U_1^1, \ldots, U_m^1 \in \mathcal{P}(D), \quad U_1^2, \ldots, U_p^2 \in \mathcal{P}(\mathcal{P}(D))$.

By $\boldsymbol{\mathcal{M}}[\vec{x}/\vec{u}, \vec{X}^1/\vec{U}^1, \vec{X}^2/\vec{U}^2]$, we denote the interpretation $\boldsymbol{\mathcal{M}}' = (D, M')$ such that $M'x_i = u_i$ (for $i = 1, \ldots, n$), $M'X_j^1 = U_j^1$ (for $j = 1, \ldots, m$), $M'X_k^2 = U_k^2$ (for $k = 1, \ldots, p$), and which otherwise coincides with M on all remaining variables. For a 4LQS-interpretation $\boldsymbol{\mathcal{M}} = (D, M)$ and a 4LQS-formula φ, the satisfiability relationship $\boldsymbol{\mathcal{M}} \models \varphi$ is defined inductively over the structure of φ as follows. Quantifier-free atomic formulae are evaluated in a standard way according to the usual meaning of the predicates '$\in$' and '$=$', and purely universal formulae are evaluated as follows:

- $\boldsymbol{\mathcal{M}} \models (\forall z_1) \ldots (\forall z_n)\varphi_0$ iff $\boldsymbol{\mathcal{M}}[\vec{z}/\vec{u}] \models \varphi_0$, for all $\vec{u} \in D^n$;
- $\boldsymbol{\mathcal{M}} \models (\forall Z_1^1) \ldots (\forall Z_m^1)\varphi_1$ iff $\boldsymbol{\mathcal{M}}[\vec{Z}^1/\vec{U}^1] \models \varphi_1$, for all $\vec{U}^1 \in \big(\mathcal{P}(D)\big)^m$;
- $\boldsymbol{\mathcal{M}} \models (\forall Z_1^2) \ldots (\forall Z_p^2)\varphi_2$ iff $\boldsymbol{\mathcal{M}}[\vec{Z}^2/\vec{U}^2] \models \varphi_2$, for all $\vec{U}^2 \in \big(\mathcal{P}(\mathcal{P}(D))\big)^p$.

Finally, compound formulae are interpreted according to the standard rules of propositional logic. If $\boldsymbol{\mathcal{M}} \models \varphi$, then $\boldsymbol{\mathcal{M}}$ is said to be a 4LQS-model for φ. A 4LQS-formula is said to be *satisfiable* if it has a 4LQS-model. A 4LQS-formula is *valid* if it is satisfied by all 4LQS-interpretations.

We are now ready to present the fragment $\mathsf{4LQS}^{\mathsf{R}}$ of 4LQS of our interest. This is the collection of the formulae ψ of 4LQS fulfilling the restrictions:

1. for every purely universal formula $(\forall Z_1^1) \ldots (\forall Z_m^1)\varphi_1$ of level 2 occurring in ψ and every purely universal formula $(\forall z_1) \ldots (\forall z_n)\varphi_0$ of level 1 occurring negatively in φ_1, φ_0 is a propositional combination of quantifier-free atomic formulae of level 0 and the condition
$$\neg\varphi_0 \rightarrow \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} z_i \in Z_j^1$$
is a valid 4LQS-formula (in this case we say that $(\forall z_1) \ldots (\forall z_n)\varphi_0$ is *linked to the variables* $Z_1^1, \ldots, Z_m^1$);
2. for every purely universal formula $(\forall Z_1^2) \ldots (\forall Z_p^2)\varphi_2$ of level 3 in ψ:

- every purely universal formula of level 1 occurring negatively in φ_2 and not occurring in a purely universal formula of level 2 is only allowed to be of the form
$$(\forall z_1)\dots(\forall z_n)\neg(\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\langle z_i, z_j\rangle = Y_{ij}^2),$$
with $Y_{ij}^2 \in \mathcal{V}^2$, for $i, j = 1, \dots, n$;
- purely universal formulae $(\forall Z_1^1)\dots(\forall Z_m^1)\varphi_1$ of level 2 may occur only positively in φ_2.

Restriction 1 has been introduced for technical reasons concerning the decidability of the satisfiability problem for the fragment, while restriction 2 allows one to define binary relations and several operations on them (for space reasons details are not included here but can be found in [2]).

The semantics of $\mathsf{4LQS^R}$ plainly coincides with that of $\mathsf{4LQS}$.

2.2 The logic $\mathcal{DL}\langle\mathbf{4LQS^{R,\times}}\rangle(\mathbf{D})$

The description logic $\mathcal{DL}\langle\mathsf{4LQS^{R,\times}}\rangle(\mathbf{D})$ (more simply referred to as $\mathcal{DL}_{\mathbf{D}}^{4,\times}$) is the extension of the logic $\mathcal{DL}\langle\mathsf{4LQS^R}\rangle(\mathbf{D})$ (for short $\mathcal{DL}_{\mathbf{D}}^{4}$) presented in [7] in which Boolean operations on concrete roles and the product of concepts are admitted. Analogously to $\mathcal{DL}_{\mathbf{D}}^{4}$, the logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ supports concept constructs such as full negation, union and intersection of concepts, concept domain and range, existential quantification and min cardinality on the left-hand side of inclusion axioms, role constructs such as role chains on the left hand side of inclusion axioms, union, intersection, and complement of roles, and properties on roles such as transitivity, symmetry, reflexivity, and irreflexivity.

As far as the construction of role inclusion axioms is concerned, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is more liberal than $\mathcal{SROIQ}(\mathbf{D})$ [12] (the logic underlying the most expressive Ontology Web Language 2 profile, OWL 2 DL [16]), since the roles involved are not required to be subject to any ordering relationship, and the notion of simple role is not needed. $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ treats derived datatypes by admitting datatype terms constructed from data ranges by means of a finite number of applications of the Boolean operators. Basic and derived datatypes can be used inside inclusion axioms involving concrete roles.

Datatypes are defined according to [14] as follows. Let $\mathbf{D} = (N_D, N_C, N_F, \cdot^{\mathbf{D}})$ be a *datatype map*, where N_D is a finite set of datatypes, N_C is a map assigning a set of constants $N_C(d)$ to each datatype $d \in N_D$, N_F is a map assigning a set of facets $N_F(d)$ to each $d \in N_D$, and $\cdot^{\mathbf{D}}$ is a map assigning (i) a datatype interpretation $d^{\mathbf{D}}$ to each datatype $d \in N_D$, (ii) a facet interpretation $f^{\mathbf{D}} \subseteq d^{\mathbf{D}}$ to each facet $f \in N_F(d)$, and (iii) a data value $e_d^{\mathbf{D}} \in d^{\mathbf{D}}$ to every constant $e_d \in N_C(d)$. We shall assume that the interpretations of the datatypes in N_D are non-empty pairwise disjoint sets.

A *facet expression* for a datatype $d \in N_D$ is a formula ψ_d constructed from the elements of $N_F(d) \cup \{\top_d, \bot_d\}$ by applying a finite number of times the connectives $\neg$, $\wedge$, and $\vee$. The function $\cdot^{\mathbf{D}}$ is extended to facet expressions for

$d \in N_D$ by putting $\top_d^{\mathbf{D}} = d^{\mathbf{D}}$, $\bot_d^{\mathbf{D}} = \emptyset$, $(\neg f)^{\mathbf{D}} = d^{\mathbf{D}} \setminus f^{\mathbf{D}}$, $(f_1 \wedge f_2)^{\mathbf{D}} = f_1^{\mathbf{D}} \cap f_2^{\mathbf{D}}$, and $(f_1 \vee f_2)^{\mathbf{D}} = f_1^{\mathbf{D}} \cup f_2^{\mathbf{D}}$, for $f, f_1, f_2 \in N_F(d)$.

A *data range dr* for $\mathbf{D}$ is either a datatype $d \in N_D$, or a finite enumeration of datatype constants $\{e_{d_1}, \ldots, e_{d_n}\}$, with $e_{d_i} \in N_C(d_i)$ and $d_i \in N_D$, or a facet expression ψ_d, for $d \in N_D$, or their complementation.

Let $\mathbf{R_A}$, $\mathbf{R_D}$, $\mathbf{C}$, $\mathbf{Ind}$ be denumerable pairwise disjoint sets of abstract role names, concrete role names, concept names, and individual names, respectively. We assume that the set of abstract role names $\mathbf{R_A}$ contains a name U denoting the universal role.
(a) $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-datatype, (b) $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept, (c) $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role, and (d) $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role terms are constructed according to the following syntax rules:

(a) $t_1, t_2 \longrightarrow dr \mid \neg t_1 \mid t_1 \sqcap t_2 \mid t_1 \sqcup t_2 \mid \{e_d\}$,
(b) $C_1, C_2 \longrightarrow A \mid \top \mid \bot \mid \neg C_1 \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \{a\} \mid \exists R.Self \mid \exists R.\{a\} \mid \exists P.\{e_d\}$,
(c) $R_1, R_2 \longrightarrow S \mid U \mid R_1^- \mid \neg R_1 \mid R_1 \sqcup R_2 \mid R_1 \sqcap R_2 \mid R_{C_1|} \mid R_{|C_1} \mid R_{C_1 \mid C_2} \mid id(C) \mid C_1 \times C_2$,
(d) $P_1, P_2 \longrightarrow T \mid \neg P_1 \mid P_1 \sqcup P_2 \mid P_1 \sqcap P_2 \mid P_{C_1|} \mid P_{|t_1} \mid P_{C_1|t_1}$,

where dr is a data range for $\mathbf{D}$, t_1, t_2 are data-type terms, e_d is a constant in $N_C(d)$, a is an individual name, A is a concept name, C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, S is an abstract role name, R, R_1, R_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms, T is a concrete role name, and P, P_1, P_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role terms.

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base is a triple $\mathcal{KB} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ such that $\mathcal{R}$ is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*RBox*, $\mathcal{T}$ is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*TBox*, and $\mathcal{A}$ a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*ABox* (see next).

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*RBox* is a collection of statements of the following forms:

$R_1 \equiv R_2$, $R_1 \sqsubseteq R_2$, $R_1 \ldots R_n \sqsubseteq R_{n+1}$, $\mathsf{Sym}(R_1)$, $\mathsf{Asym}(R_1)$, $\mathsf{Ref}(R_1)$, $\mathsf{Irref}(R_1)$, $\mathsf{Dis}(R_1, R_2)$, $\mathsf{Tra}(R_1)$, $\mathsf{Fun}(R_1)$, $R_1 \equiv C_1 \times C_2$, $P_1 \equiv P_2$, $P_1 \sqsubseteq P_2$, $\mathsf{Dis}(P_1, P_2)$, $\mathsf{Fun}(P_1)$, where R_1, R_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms, C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract concept terms, and P_1, P_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role terms. Any expression of the type $w \sqsubseteq R$, where w is a finite string of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms and R is an $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term is called a *role inclusion axiom (RIA)*.

Next, a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*TBox* is a set of statements of the types:

$C_1 \equiv C_2$, $C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \forall R_1.C_2$, $\exists R_1.C_1 \sqsubseteq C_2$, $\geq_n R_1.C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \leq_n R_1.C_2$,
$t_1 \equiv t_2$, $t_1 \sqsubseteq t_2$, $C_1 \sqsubseteq \forall P_1.t_1$, $\exists P_1.t_1 \sqsubseteq C_1$, $\geq_n P_1.t_1 \sqsubseteq C_1$, $C_1 \sqsubseteq \leq_n P_1.t_1$,

where C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, t_1, t_2 datatype terms, R_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, and P_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term. Any statement $C \sqsubseteq D$, with C and D $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, is a *general concept inclusion axiom (GCI)*.

Finally, a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*ABox* is a set of *individual assertions* of the forms: $a : C_1$, $(a, b) : R_1$, $a = b$, $e_d : t_1$, $(a, e_d) : P_1$, with a, b individual names, C_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept term, R_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, d a datatype, e_d a constant in $N_C(d)$, t_1 a datatype term, and P_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term. As mentioned above, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is more liberal than $\mathcal{SROIQ}(\mathbf{D})$ in the construction of role inclusion axioms. For example, the role hierarchy $\{RS \sqsubseteq S, RT \sqsubseteq R, VT \sqsubseteq T, VS \sqsubseteq V\}$ presented in [12] is expressible in $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, but not in $\mathcal{SROIQ}(\mathbf{D})$.

The semantics of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is based on interpretations $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$, where $\Delta^{\mathbf{I}}$ and $\Delta_{\mathbf{D}}$ are non-empty disjoint domains such that $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, for every $d \in N_D$, and $\cdot^{\mathbf{I}}$ is an interpretation function. The interpretation of concepts and roles, and of axioms and assertions is illustrated in [3, Table 1].

Let $\mathcal{KB} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base. An interpretation $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$ is a $\mathbf{D}$-model of $\mathcal{R}$ (and write $\mathbf{I} \models_{\mathbf{D}} \mathcal{R}$) if $\mathbf{I}$ satisfies each axiom in $\mathcal{R}$ according to the semantic rules in [3, Table 1]. Similar definitions hold for $\mathcal{T}$ and $\mathcal{A}$ too. Then $\mathbf{I}$ satisfies $\mathcal{KB}$ (and write $\mathbf{I} \models_{\mathbf{D}} \mathcal{KB}$) if it is a $\mathbf{D}$-model of $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{A}$. A knowledge base is *consistent* if it is satisfied by some interpretation.

3 Conjunctive Query Answering for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$

Let $\mathcal{V} = \{v_1, v_2, \ldots\}$ be a denumerable and infinite set of variables disjoint from $\mathbf{Ind}$ and from $\bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$. A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*atomic formula* is an expression of of the following types

$$R(w_1, w_2), \quad P(w_1, u_1), \quad C(w_1), \quad w_1 = w_2, \quad u_1 = u_2,$$

where $w_1, w_2 \in \mathcal{V} \cup \mathbf{Ind}$, $u_1, u_2 \in \mathcal{V} \cup \bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$, R is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, P is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term, and C is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept term. A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-atomic formula containing no variables is said to be *closed*. A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*literal* is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-atomic formula or its negation. A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*conjunctive query* is a conjunction of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-literals. Let $v_1, \ldots, v_n \in \mathcal{V}$ and $o_1, \ldots, o_n \in \mathbf{Ind} \cup \bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$. A substitution $\sigma \coloneqq \{v_1/o_1, \ldots, v_n/o_n\}$ is a map such that, for every $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-literal L, $L\sigma$ is obtained from L by replacing the occurrences of $v_1, \ldots, v_n$ in L with $o_1, \ldots, o_n$, respectively. Substitutions can be extended to $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries in the usual way. Let $Q \coloneqq (L_1 \wedge \ldots \wedge L_m)$ be a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query, and $\mathcal{KB}$ a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base. A substitution σ involving *exactly* the variables occurring in Q is a *solution for Q w.r.t. $\mathcal{KB}$* if there exists a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-interpretation $\mathbf{I}$ such that $\mathbf{I} \models_{\mathbf{D}} \mathcal{KB}$ and $\mathbf{I} \models_{\mathbf{D}} Q\sigma$. The collection Σ of the solutions for Q w.r.t. $\mathcal{KB}$ is the *answer set of Q w.r.t. $\mathcal{KB}$*. Then the *conjunctive query answering* (CQA) problem for Q w.r.t. $\mathcal{KB}$ consists in finding the answer set Σ of Q w.r.t. $\mathcal{KB}$.

We shall solve the CQA problem just stated by reducing it to the analogous problem formulated in the context of the fragment $\mathsf{4LQS^R}$ (and in turn to the decision procedure for $\mathsf{4LQS^R}$ presented in [2]). The CQA problem for $\mathsf{4LQS^R}$-formulae can be stated as follows. Let ϕ be a $\mathsf{4LQS^R}$-formula and let ψ be a conjunction of $\mathsf{4LQS^R}$-quantifier-free atomic formulae of level 0 of the types $x = y$, $x \in X^1$, $\langle x, y\rangle \in X^3$ or their negations, such that $\mathtt{Var}_0(\psi) \cap \mathtt{Var}_0(\phi) = \emptyset$ and $\mathtt{Var}_1(\psi) \cup \mathtt{Var}_3(\psi) \subseteq \mathtt{Var}_1(\phi) \cup \mathtt{Var}_3(\phi)$. The *CQA problem for ψ w.r.t. ϕ* consists in computing the *answer set of ψ w.r.t. ϕ*, namely the collection Σ' of all the substitutions $\sigma' \coloneqq \{x_1/y_1, \ldots, x_n/y_n\}$ (where $x_1, \ldots, x_n$ are the distinct variables of level 0 in ψ and $\{y_1, \ldots, y_n\} \subseteq \mathtt{Var}_0(\phi)$) such that $\boldsymbol{\mathcal{M}} \models \phi \wedge \psi\sigma'$, for some $\mathsf{4LQS^R}$-interpretation $\boldsymbol{\mathcal{M}}$. In view of the decidability of the satisfiability problem for $\mathsf{4LQS^R}$-formulae, the CQA problem for $\mathsf{4LQS^R}$-formulae is decidable as well. Indeed, given two $\mathsf{4LQS^R}$-formulae ϕ and ψ satisfying the

above requirements, to compute the answer set of ψ w.r.t. ϕ, for each candidate substitution $\sigma' \coloneqq \{x_1/y_1, \ldots, x_n/y_n\}$ (with $\{x_1, \ldots, x_n\} = \mathtt{Var}_0(\psi)$ and $\{y_1, \ldots, y_n\} \subseteq \mathtt{Var}_0(\phi)$) one has just to test for satisfiability the $\mathsf{4LQS^R}$-formula $\phi \wedge \psi\sigma'$. Since the number of possible candidate substitutions is $|\mathtt{Var}_0(\phi)|^{|\mathtt{Var}_0(\psi)|}$ and the satisfiability problem for $\mathsf{4LQS^R}$-formulae is the decidable, it follows that the answer set of ψ w.r.t. ϕ can be computed effectively. Summarizing,

Lemma 1. *The CQA problem for $\mathsf{4LQS^R}$-formulae is decidable.* □

The following theorem states that also the CQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is decidable.

Theorem 1. *Given a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base $\mathcal{KB}$ and a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query Q, the CQA problem for Q w.r.t. $\mathcal{KB}$ is decidable.*

Proof (sketch). For space reasons we just outline the main ideas of the proof. The interested reader, however, can find complete details in the extended version of this paper (see [3]).

As remarked above, the CQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ can be solved via an effective reduction to the CQA problem for $\mathsf{4LQS^R}$-formulae, and then exploiting Lemma 1. The reduction is accomplished through a function θ that maps effectively variables in $\mathcal{V}$ and individuals in **Ind** into variables of sort 0 (of the $\mathsf{4LQS^R}$-language), etc., $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*TBoxes*, -*RBoxes*, and -*ABoxes*, and $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries into $\mathsf{4LQS^R}$-formulae in conjunctive normal form (CNF), which can be used to map effectively CQA problems from the $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-context into the $\mathsf{4LQS^R}$-context. More specifically, given a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base $\mathcal{KB}$ and a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query Q, using the function θ we can effectively construct the following $\mathsf{4LQS^R}$-formulae in CNF:

$$\phi_{\mathcal{KB}} \coloneqq \bigwedge_{H \in \mathcal{KB}} \theta(H) \wedge \bigwedge_{i=1}^{12} \xi_i, \qquad \psi_Q \coloneqq \theta(Q)\,.^2$$

Then, if we denote by Σ the answer set of Q w.r.t. $\mathcal{KB}$ and by Σ' the answer set of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$, we have that Σ consists of all substitutions σ (involving exactly the variables occurring in Q) such that $\theta(\sigma) \in \Sigma'$. Since, by Lemma 1, Σ' can be computed effectively, then Σ can be computed effectively too. □

4 A tableau-based procedure

In this section, we illustrate a KE-tableau based procedure that, given a $\mathsf{4LQS^R}$-formula $\phi_{\mathcal{KB}}$ corresponding to a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base and a $\mathsf{4LQS^R}$-formula

[2] The definition of the function θ is inspired to that of the function τ introduced in the proof of Theorem 1 in [7]. Specifically, θ differs from τ as (i) it allows quantification only on variables of level 0, (ii) it treats Boolean operations on concrete roles and the product of concepts, and (iii) it constructs $\mathsf{4LQS^R}$-formulae in CNF. In addition, the constraints ξ_1–ξ_{12} are similar to the constraints ψ_1–ψ_{12} introduced in the proof of Theorem 1 in [7]; they are introduced to guarantee that each model of $\phi_{\mathcal{KB}}$ can be transformed into a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-interpretation. Details of the construction of θ and of ξ_1–ξ_{12} can be found in [3].

ψ_Q corresponding to a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query Q, yields all the substitutions $\sigma = \{x_1/y_1, \ldots, x_n/y_n\}$, with $\{x_1, \ldots, x_n\} = \mathtt{Var}_0(\psi_Q)$ and $\{y_1, \ldots, y_n\} \subseteq \mathtt{Var}_0(\phi_{\mathcal{KB}})$, belonging to the answer set Σ' of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$.

Let $\overline{\phi}_{\mathcal{KB}}$ be the formula obtained from $\phi_{\mathcal{KB}}$ by:

- moving universal quantifiers in $\phi_{\mathcal{KB}}$ as inwards as possible according to the rule $(\forall z)(A(z) \wedge B(z)) \longleftrightarrow ((\forall z)A(z) \wedge (\forall z)B(z))$,
- renaming universally quantified variables so as to make them pairwise distinct.

Let $F_1, \ldots, F_k$ be the conjuncts of $\overline{\phi}_{\mathcal{KB}}$ that are $\mathsf{4LQS^R}$-quantifier-free atomic formulae and $S_1, \ldots, S_m$ the conjuncts of $\overline{\phi}_{\mathcal{KB}}$ that are $\mathsf{4LQS^R}$-purely universal formulae. For every $S_i = (\forall z_1^i) \ldots (\forall z_{n_i}^i)\chi_i$, $i = 1, \ldots, m$, we put

$$Exp(S_i) \coloneqq \bigwedge_{\{x_{a_1}, \ldots, x_{a_{n_i}}\} \subseteq \mathtt{Var}_0(\overline{\phi}_{\mathcal{KB}})} S_i\{z_1^i/x_{a_1}, \ldots, z_{n_i}^i/x_{a_{n_i}}\}.$$

Let $\Phi_{\mathcal{KB}} \coloneqq \{F_j : i = 1, \ldots, k\} \cup \bigcup_{i=1}^{m} Exp(S_i)$.

To prepare for the KE-tableau based procedure to be described next, we introduce some useful notions and notations (see [10] for a detailed overview of KE-tableau, an optimized variant of semantic tableaux).

Let $\Phi = \{C_1, \ldots, C_p\}$ be a collection of disjunctions of $\mathsf{4LQS^R}$-quantifier-free atomic formulae of level 0 of the types: $x = y$, $x \in X^1$, $\langle x, y\rangle \in X^3$. $\mathcal{T}$ is a *KE-tableau* for Φ if there exists a finite sequence $\mathcal{T}_1, \ldots, \mathcal{T}_t$ such that (i) $\mathcal{T}_1$ is a one-branch tree consisting of the sequence $C_1, \ldots, C_p$, (ii) $\mathcal{T}_t = \mathcal{T}$, and (iii) for each $i < t$, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by an application of one of the rules in Fig 1. The set of formulae $\mathcal{S}_i^{\overline{\beta}} = \{\overline{\beta}_1, \ldots, \overline{\beta}_n\} \setminus \{\overline{\beta}_i\}$ occurring as premise in the E-rule contains the complements of all the components of the formula β with the exception of the component β_i.

$$\frac{\beta_1 \vee \ldots \vee \beta_n \qquad \mathcal{S}_i^{\overline{\beta}}}{\beta_i}\ \textbf{E-Rule} \qquad\qquad \frac{}{A \mid \overline{A}}\ \textbf{PB-Rule}$$

where $\mathcal{S}_i^{\overline{\beta}} \coloneqq \{\overline{\beta}_1, \ldots, \overline{\beta}_n\} \setminus \{\overline{\beta}_i\}$, for $i = 1, \ldots, n$

with A a literal

Fig. 1. Expansion rules for the KE-tableau.

Let $\mathcal{T}$ be a KE-tableau. A branch ϑ of $\mathcal{T}$ is *closed* if it contains both A and $\neg A$, for some formula A. Otherwise, the branch is *open*. A formula $\beta_1 \vee \ldots \vee \beta_n$ is *fulfilled* in a branch ϑ, if β_i is in ϑ, for some $i = 1, \ldots, n$. A branch ϑ is *complete* if every formula $\beta_1 \vee \ldots \vee \beta_n$ occurring in ϑ is fulfilled. A KE-tableau is *complete* if all its branches are complete.

Next we introduce the procedure Saturate-KB that takes as input the set $\Phi_{\mathcal{KB}}$ constructed from a $\mathsf{4LQS^R}$-formula $\phi_{\mathcal{KB}}$ representing a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base $\mathcal{KB}$ as shown above, and yields a complete KE-tableau $\mathcal{T}_{\mathcal{KB}}$ for $\Phi_{\mathcal{KB}}$.

Procedure 1 *Saturate-KB($\Phi_{\mathcal{KB}}$)*

1. $\mathcal{T}_{\mathcal{KB}} := \Phi_{\mathcal{KB}}$;

2. *Select an open branch ϑ of $\mathcal{T}_{\mathcal{KB}}$ that is not yet complete.*
   (a) *Select a formula $\beta_1 \vee \ldots \vee \beta_n$ on ϑ that is not fulfilled.*
   (b) *If $\mathcal{S}_j^{\overline{\beta}}$ is in ϑ, for some $j \in \{1, \ldots, n\}$, apply the E-Rule to $\beta_1 \vee \ldots \vee \beta_n$ and $\mathcal{S}_j^{\overline{\beta}}$ on ϑ and go to step 2.*
   (c) *If $\mathcal{S}_j^{\overline{\beta}}$ is not in ϑ, for every $j = 1, \ldots, n$, let $B^{\overline{\beta}}$ be the collection of formulae $\overline{\beta}_1, \ldots, \overline{\beta}_n$ present in ϑ and let $\overline{\beta}_h$ be the lowest index formula such that $\overline{\beta}_h \in \{\{\overline{\beta}_1, \ldots, \overline{\beta}_n\} \setminus B^{\overline{\beta}}\}$, then apply the PB-rule to $\overline{\beta}_h$ on ϑ, and go to step 2.*
3. *Return $\mathcal{T}_{\mathcal{KB}}$.*

Soundness of Procedure 1 can be easily proved in a standard way and its completeness can be shown much along the lines of Proposition 36 in [10]. Concerning termination of Procedure 1, our proof is based on the following two facts. The rules in Fig. 1 are applied only to non-fulfilled formulae on open branches and tend to reduce the number of non-fulfilled formulae occurring on the considered branch. In particular, when the E-Rule is applied on a branch ϑ, the number of non-fulfilled formulae on ϑ decreases. In case of application of the PB-Rule on a formula $\beta = \beta_1 \vee \ldots \vee \beta_n$ on a branch, the rule generates two branches. In one of them the number of non-fulfilled formulae decreases (because β becomes fulfilled). In the other one the number of non-fulfilled formulae stays constant but the subset $B^{\overline{\beta}}$ of $\{\overline{\beta}_1, \ldots, \overline{\beta}_n\}$ occurring on the branch gains a new element. Once $|B^{\overline{\beta}}|$ gets equal to $n-1$, namely after at most $n-1$ applications of the PB-rule, the E-rule is applied and the formula $\beta = \beta_1 \vee \ldots \vee \beta_n$ becomes fulfilled, thus decrementing the number of non-fulfilled formulae on the branch. Since the number of non-fulfilled formulae on each open branch gets equal to zero after a finite number of steps and the rules of Fig. 1 can be applied only to non-fulfilled formulae on open branches, the procedure terminates.

By the completeness of Procedure 1, each open branch ϑ of $\mathcal{T}_{\mathcal{KB}}$ induces a $\mathsf{4LQS^R}$-interpretation $\boldsymbol{\mathcal{M}}_\vartheta$ such that $\boldsymbol{\mathcal{M}}_\vartheta \models \Phi_{\mathcal{KB}}$. We define $\boldsymbol{\mathcal{M}}_\vartheta = (D_\vartheta, M_\vartheta)$ as follows. We put $D_\vartheta := \{x \in \mathcal{V}_0 : x \text{ occurs in } \vartheta\}$; $M_\vartheta x := x$, for every $x \in D_\vartheta$; $M_\vartheta X_C^1 = \{x : x \in X_C^1 \text{ is in } \vartheta\}$, for every $X_C^1 \in \mathcal{V}_1$ occurring ϑ; $M_\vartheta X_R^3 = \{\langle x, y\rangle : \langle x, y\rangle \in X_R^3 \text{ is in } \vartheta\}$, for every $X_R^3 \in \mathcal{V}_3$ occurring in ϑ. It is easy to check that $\boldsymbol{\mathcal{M}}_\vartheta \models \overline{\phi}_{\mathcal{KB}}$ and thus, plainly, that $\boldsymbol{\mathcal{M}}_\vartheta \models \phi_{\mathcal{KB}}$.

Next, we provide some complexity results. Let r be the maximum number of universal quantifiers in S_i, and $k := |\mathtt{Var}_0(\overline{\phi}_{\mathcal{KB}})|$. Then, each S_i generates k^r expansions. Since the knowledge base contains m such formulae, the number of disjunctions in the initial branch of the KE-tableau is $m \cdot k^r$. Next, let ℓ be the maximum number of literals in S_i, for $i = 1, \ldots, m$. Then, the maximum depth of the KE-tableau, namely the maximum size of the models of $\Phi_{\mathcal{KB}}$ constructed as illustrated above, is $\mathcal{O}(\ell m k^r)$ and the number of leaves of the tableau, that is the number of such models of $\Phi_{\mathcal{KB}}$, is $O(2^{\ell m k^r})$.

We now describe a procedure that, given a KE-tableau constructed by Procedure 1 and a $\mathsf{4LQS^R}$-formula ψ_Q representing a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query Q, yields all the substitutions σ' in the answer set Σ' of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$. By the soundness of Procedure 1, we can limit ourselves to consider only the models $\boldsymbol{\mathcal{M}}_\vartheta$ of $\phi_{\mathcal{KB}}$ induced by each open branch ϑ of $\mathcal{T}_{\mathcal{KB}}$. For every open and complete branch ϑ

of $\mathcal{T}_{\mathcal{KB}}$, we construct a decision tree $\mathcal{D}_\vartheta$ such that every maximal branch of $\mathcal{D}_\vartheta$ defines a substitution σ' such that $\boldsymbol{\mathcal{M}}_\vartheta \models \psi_Q\sigma'$.

Let d be the number of literals in ψ_Q. $\mathcal{D}_\vartheta$ is a finite labelled tree of depth $d+1$ whose labelling satisfies the following conditions, for $i = 0, \ldots, d$: (i) every node of $\mathcal{D}_\vartheta$ at level i is labelled with $(\sigma_i, \psi_Q\sigma_i)$, and, in particular, the root is labelled with $(\sigma'_0, \psi_Q\sigma'_0)$, where σ'_0 is the empty substitution; (ii) if a node at level i is labelled with $(\sigma'_i, \psi_Q\sigma'_i)$, then its s-successors, with $s > 0$, are labelled with $\big(\sigma'_i\varrho_1^{q_i+1}, \psi_Q(\sigma'_i\varrho_1^{q_i+1})\big), \ldots, \big(\sigma'_i\varrho_s^{q_i+1}, \psi_Q(\sigma'_i\varrho_s^{q_i+1})\big)$, where q_{i+1} is the $(i+1)$-st conjunct of $\psi_Q\sigma'_i$ and $\mathcal{S}_{q_{i+1}} = \{\varrho_1^{q_i+1}, \ldots, \varrho_s^{q_i+1}\}$ is the collection of the substitutions $\varrho = \{x_1/y_1, \ldots, x_j/y_j\}$ with $\{x_1, \ldots, x_j\} = \mathtt{Var}_0(q_{i+1})$ such that $p = q_{i+1}\varrho$, for some literal p on ϑ. If $s = 0$, the node labelled with $(\sigma'_i, \psi_Q\sigma'_i)$ is a leaf node and, if $i = d$, σ'_i is added to Σ'.

Let $\delta(\mathcal{T}_{\mathcal{KB}})$ and $\lambda(\mathcal{T}_{\mathcal{KB}})$ be, respectively, the maximum depth of $\mathcal{T}_{\mathcal{KB}}$ and the number of leaves of $\mathcal{T}_{\mathcal{KB}}$ computed above. Plainly, $\delta(\mathcal{T}_{\mathcal{KB}}) = \mathcal{O}(\ell m k^r)$ and $\lambda(\mathcal{T}_{\mathcal{KB}}) = \mathcal{O}(2^{\ell m k^r})$. It is easy to verify that $s = 2^k$ is the maximum branching of $\mathcal{D}_\vartheta$. Since $\mathcal{D}_\vartheta$ is a s-ary tree of depth $d+1$, where d is the number of literals in ψ_Q, and the s-successors of a node are computed in $\mathcal{O}(\delta(\mathcal{T}_{\mathcal{KB}}))$ time, the number of leaves in $\mathcal{D}_\vartheta$ is $\mathcal{O}(s^{(d+1)}) = \mathcal{O}(2^{k(d+1)})$ and they are computed in $\mathcal{O}(2^{k(d+1)}\delta(\mathcal{T}_{\mathcal{KB}}))$ time. Finally, since we have $\lambda(\mathcal{T}_{\mathcal{KB}})$ of such decision trees, the answer set of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$ is computed in $\mathcal{O}(2^{k(d+1)}\delta(\mathcal{T}_{\mathcal{KB}})\lambda(\mathcal{T}_{\mathcal{KB}})) = \mathcal{O}(2^{k(d+1)} \cdot \ell m k^r \cdot 2^{\ell m k^r}) = \mathcal{O}(\ell m k^r 2^{k(d+1)+\ell m k^r})$ time. Since the size of $\phi_{\mathcal{KB}}$ and of ψ_Q are polynomially related to those of $\mathcal{KB}$ and of Q, respectively (see [3] for details), the construction of the answer set of Q with respect to $\mathcal{KB}$ can be done in double-exponential time. In case $\mathcal{KB}$ contains no role chain axioms and qualified cardinality restrictions, the complexity of our CQA problem is in EXPTIME, since the maximum number of universal quantifiers in $\phi_{\mathcal{KB}}$, namely r, is a constant (in particular $r = 3$). We remark that such result is comparable with the complexity of the CQA problem for a large family of description logics such as $\mathcal{SHIQ}$ [15]. In particular, the CQA problem for the very expressive description logic $\mathcal{SROIQ}$ turns out to be 2-NEXPTIME-complete.

5 Conclusions

We have introduced the description logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R},\times}\rangle(\mathbf{D})$ ($\mathcal{DL}_{\mathbf{D}}^{4,\times}$, for short) that extends the logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R}}\rangle(\mathbf{D})$ with Boolean operations on concrete roles and with the product of concepts. We addressed the problem of Conjunctive Query Answering for the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ by formalizing $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge bases and $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries in terms of formulae of $\mathsf{4LQS}^{\mathsf{R}}$. Such formalization seems to be promising for implementation purposes.

In our approach, we first constructed a KE-tableau $\mathcal{T}_{\mathcal{KB}}$ for $\phi_{\mathcal{KB}}$, a $\mathsf{4LQS}^{\mathsf{R}}$-formalization of a given $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge base $\mathcal{KB}$, whose branches induce the models of $\phi_{\mathcal{KB}}$. Then we computed the answer set of a $\mathsf{4LQS}^{\mathsf{R}}$-formula ψ_Q, representing a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query Q, with respect to $\phi_{\mathcal{KB}}$ by means of a forest of decision trees based on the branches of $\mathcal{T}_{\mathcal{KB}}$ and gave some complexity results.

We plan to generalize our procedure with a data-type checker in order to extend reasoning with data-types, and also to extend $\mathsf{4LQS}^{\mathsf{R}}$ with data-type groups. We also intend to improve the efficiency of the knowledge base saturation algorithm and query answering algorithm, and to extend the expressiveness of the queries. Finally, we intend to study a parallel model of the procedure described and to provide an implementation of it.

References

1. D. Calvanese, G. D. Giacomo, and M. Lenzerini, "On the decidability of query containment under constraints," in *Proc. of the* 17^{th} *ACM SIGACT-SIGMOD-SIGART Symp. on Princ. of Database Systems*, ser. PODS '98. New York, NY, USA: ACM, 1998, pp. 149–158.
2. D. Cantone and M. Nicolosi-Asmundo, "On the satisfiability problem for a 4-level quantified syllogistic and some applications to modal logic," *Fundamenta Informaticae*, vol. 124, no. 4, pp. 427–448, 2013.
3. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria, "Conjunctive Query Answering via a Fragment of Set Theory (Extended Version)," *CoRR*, vol. abs/1606.07337, 2016.
4. D. Cantone and C. Longo, "A decidable two-sorted quantified fragment of set theory with ordered pairs and some undecidable extensions," *Theor. Comput. Sci.*, vol. 560, pp. 307–325, 2014.
5. D. Cantone, C. Longo, and M. Nicolosi-Asmundo, "A decision procedure for a two-sorted extension of multi-level syllogistic with the Cartesian product and some map constructs," in *Proc. of the 25th Italian Conference on Computational Logic (CILC 2010), Rende, Italy, July 7-9, 2010*, W. Faber and N. Leone, Eds., vol. 598. CEUR Workshop Proceedings, ISSN 1613-0073, June 2010, pp. 1–18 (paper 11).
6. D. Cantone, C. Longo, and M. Nicolosi-Asmundo, "A decidable quantified fragment of set theory involving ordered pairs with applications to description logics," in *Proc. Computer Science Logic, 20th Annual Conf. of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway*, 2011, pp. 129–143.
7. D. Cantone, C. Longo, M. Nicolosi-Asmundo, and D. F. Santamaria, "Web ontology representation and reasoning via fragments of set theory," in *Web Reasoning and Rule Systems - 9th International Conference, RR 2015, Berlin, Germany, August 4-5, 2015, Proc.*, LNCS vol. 9209, Springer, ISBN 978-3-319-22001-7 pp. 61–76.
8. D. Cantone, C. Longo, and A. Pisasale, "Comparing description logics with multi-level syllogistics: the description logic $\mathcal{DL}\langle\mathsf{MLSS}^{\times}_{2,m}\rangle$," in *6th Workshop on Semantic Web Applications and Perspectives (Bressanone, Italy, Sep. 21-22, 2010)*, P. Traverso, Ed., 2010, pp. 1–13.
9. D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, and F. Trapani "Ontoceramic: an owl ontology for ceramics classification," in *Proc. of the 30th Italian Conf. on Computational Logic, CILC 2015*, July 1-3 2015 Genova, CEUR Electronic Workshop Proceedings, vol. 1459, pp. 122–127.
10. M. D'Agostino, "Tableau methods for classical propositional logic," in *Handbook of Tableau Methods*, M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, Eds. Springer, 1999, pp. 45–123.
11. N. Dershowitz and J.-P. Jouannaud, "Rewrite systems," in *Handbook of Theoretical Computer Science (Vol. B)*, J. van Leeuwen, Ed. Cambridge, MA, USA: MIT Press, 1990, pp. 243–320.

12. I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," in *Proc. 10th Int. Conf. on Princ. of Knowledge Representation and Reasoning, (Doherty, P. and Mylopoulos, J. and Welty, C. A., eds.)*. AAAI Press, 2006, pp. 57–67.
13. U. Hustadt, B. Motik, and U. Sattler, "Data complexity of reasoning in very expressive description logics," in *IJCAI-05, Proc. of the* 19^{th} *Inter. Joint Conf. on Art. Intell., Edinburgh, Scotland, UK, July 30-August 5, 2005*, 2005, pp. 466–471.
14. B. Motik and I. Horrocks, "OWL datatypes: Design and implementation," in *Proc. of the 7th Int. Semantic Web Conference (ISWC 2008)*, ser. LNCS, vol. 5318. Springer, October 26–30 2008, pp. 307–322.
15. M. Ortiz, R. Sebastian, and M. Šimkus, "Query answering in the horn fragments of the description logics shoiq and sroiq," in *Proc. of the Twenty-Second International Joint Conference on Artificial Intelligence - Vol.Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1039–1044.
16. World Wide Web Consortium, "OWL 2 Web ontology language structural specification and functional-style syntax (second edition)."

A variant of Turing machines with no control states and its connection to bounded temporal memory

Domenico Cantone and Salvatore Cristofaro

Dipartimento di Matematica e Informatica, Università di Catania
Viale Andrea Doria, 6, I-95125 Catania, Italy
{cantone,cristofaro}@dmi.unict.it

Abstract. We present a variant with no control states of the Turing machine model of computation in which, at each computation step, the operation to be performed next is determined by the symbol currently scanned and by a bounded-length suffix of the sequence of the operations already executed on the tape. We show that our variant is Turing complete, i.e., it can simulate any (standard) Turing machine. (In fact, we shall provide a *strong* simulation which replicates the same tape configurations assumed by the simulated Turing machine, without using any additional tape symbol.) As a consequence, we argue that in order to perform general computation tasks, Turing machines do not need to memorize in their control states events arbitrarily far back in the past.

Keywords: Stateless Turing machines, Turing completeness, bounded temporal memory.

1 Introduction

The notion of control states of a Turing machine (TM) is strictly related to that of *temporal memory*, i.e., the ability to remember events occurred in the past.[1] In fact, TMs can remember actions performed in the past (i.e., the operations executed on the tape and/or the symbols read off from it) by encoding the information related to such actions within their control states. Since TMs have only finitely many control states, for each TM there is a fixed bound on the amount of information that can be encoded within control states, and hence a bound on the number of past actions that it can remember. However, despite this quantitative limitation, there is conceptually no limit on how "old" the actions

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 36–48 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

[1] In this paper, by a Turing machine we mean a deterministic two-way single-tape Turing machine with the instructions represented by quadruples, as described in [2].

remembered by control states can be, i.e., remembered actions can be *arbitrarily old*. To clarify this point, consider for instance the case of a TM that recognizes the strings η, over a 2-symbol alphabet $\{a, b\}$, whose leftmost and rightmost symbols are different. When the machine reads the rightmost symbol of η, it needs only to remember which was the leftmost symbol. Thus, only two control states suffice for this purpose: one to remember that the leftmost symbol of η was 'a' and one to remember that it was, instead, 'b'. Notice, however, that the number of actions that the machine can perform between the reading of the leftmost and of the rightmost symbols of η can be arbitrarily large, depending on the length of η, and therefore, when the TM reads the rightmost symbol of η, the previous reading of the leftmost symbol turns out to be arbitrarily old.

The capability to remember subsequences of arbitrarily old actions is a peculiarity of TMs and, more generally, of any other device which uses control states (e.g., *finite automata*). The main aim of this paper is to investigate to what extent this property of TMs is demanded for performing general computations. To this end, we introduce a *stateless* variant of TMs (namely, with no control states or, equivalently, with only one control state), named *Stateless Bounded Temporal Memory Turing Machines* (SBTMs).[2] From a purely mechanical point of view, SBTMs behave identically to standard TMs. However, at each computation step, the operation to be performed next on the tape by the scanning head of a SBTM (i.e., printing a symbol onto the scanned cell, or moving to the right or to the left of the current position) is determined by a suffix of the sequence of the operations already executed (including the symbols read by the head after these operations and, therefore, also the symbol currently scanned),[3] whose length cannot exceed a bound depending solely on the particular machine.

We shall prove that SBTMs are Turing complete, i.e., they can simulate any standard TM. In fact, we shall see that for each TM $\mathfrak{M}$ one can construct a SBTM $\mathfrak{V}$ which *faithfully* simulates $\mathfrak{M}$, in the sense that (i) $\mathfrak{V}$ does not use any additional symbol other than those used by $\mathfrak{M}$, (ii) on any input η, $\mathfrak{V}$ halts if and only if so does $\mathfrak{M}$, and (iii) when, on input η, $\mathfrak{V}$ and $\mathfrak{M}$ halt, they reach the same tape configuration, while scanning the same cell. As a consequence, we shall also show that the above mentioned ability of TMs to remember within their control states subsequences of arbitrarily old actions is not strictly required for performing general computation tasks (see Section 3.1 for a more precise statement of this fact). The latter property entails in particular a bound on the amount of sequential data items needed to be memorized during computations, which is a relevant topic, e.g., in the field of Streaming Algorithms [5].

Some Turing complete variants of stateless TMs have already been proposed in the literature; however, these act quite differently from ours. This is the case, for instance, for the stateless TM studied in [1], named JTM. The head of a JTM

[2] Notice that restricting to one the number of control states in TMs results in a decreased computational power (see [6]).

[3] In fact, since SBTMs are devoid of the additional component of the set of control states, the only way for these devices to be aware if something happened in the past, is to trace back their own computations.

spans a window of three consecutive tape cells and can thus read/write blocks of three symbols in a single move. However, motion to the left/right of the current head position can occur only by one cell at a time. It turns out that JTMs are Turing complete. Stateless variants of several computational devices other than TMs have also been investigated in the literature, mainly *stateless automata* such as *stateless restarting automata* [3], *stateless multihead automata* [4], *stateless pushdown automata* [7], etc. However, these devices, even in their original form, have lower computational power than TMs.

The paper is organized as follows. Below we briefly review some notations and terminology which will be used in the rest of the paper. Then, in Section 2, we provide the definition of SBTMs and their semantics. Subsequently, in Section 3, we prove the Turing completeness of SBTMs, based on a simple encoding of the states of a TM, and also discuss other similar encodings. Then, in Section 3.1, we introduce a class of TMs with bounded temporal memory and prove its Turing completeness (based on the Turing completeness of SBTMs), thereby showing the independence of the computational power of TMs from the property of remembering subsequences of arbitrarily old actions within their control states. Finally, we draw our conclusions in Section 4.

Basic notations and definitions. Both in the case of TMs and SBTMs, we shall assume that *tape symbols* are drawn from the set of symbols $\mathscr{S} := \{\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2, \ldots\}$, where s_0 is the blank. Often we shall write s_0 as $\square$. Additionally, we shall use the symbols in the set $\mathscr{A} := \{\curvearrowleft, \curvearrowright, \odot\}$ to denote certain active operations (as will be specified in the next section).

Throughout the paper, by a string we shall always mean a finite sequence of symbols belonging to the set $\mathscr{S} \cup \mathscr{A}$. In particular, for every $\mathscr{B} \subseteq \mathscr{S} \cup \mathscr{A}$, the set of all strings whose symbols belong to $\mathscr{B}$ will be denoted by $\mathscr{B}^*$. We shall write ε for the *empty string* and $|\alpha|$ for the *length* of the string α. Notice that we do *not* distinguish between a symbol and the string of length 1 consisting only of that symbol. The *concatenation* of two strings α and β is denoted by $\alpha.\beta$ or, more simply, by $\alpha\beta$. For every string α and each $n \geqslant 0$, α^n denotes the string of length $n|\alpha|$ consisting of the concatenation of n copies of α; thus, in particular, we put $\alpha^0 := \varepsilon$. A string α is a *suffix* (resp., *prefix*) of a string β, and in such a case we write $\alpha \sqsupseteq \beta$ (resp., $\alpha \sqsubseteq \beta$), if $\beta = \gamma\alpha$ (resp., $\beta = \alpha\gamma$), for some string γ; α is a *factor* of β, if $\beta = \lambda\alpha\rho$, for some strings λ and ρ.

Given a nonempty string α, we denote with $\mathsf{Head}(\alpha)$ (resp., $\mathsf{Last}(\alpha)$) the leftmost (resp., rightmost) symbol of α, and denote with $\mathsf{Init}(\alpha)$ (resp., $\mathsf{Tail}(\alpha)$) the string of length $|\alpha| - 1$ obtained by deleting the rightmost (resp., leftmost) symbol of α; we also put $\mathsf{Head}(\varepsilon) := \mathsf{Last}(\varepsilon) := \square$ and $\mathsf{Init}(\varepsilon) := \mathsf{Tail}(\varepsilon) := \varepsilon$.

2 SBTMs and their semantics

Informal description. A SBTM is equipped with the very same hardware components as a TM (see [2]), namely, a *linear tape*, infinite in both directions, divided into *cells*, with a *head* that at any given instant of time is positioned

over a particular cell (the *scanned cell*), and a finite *control box* that determines the operations that the head performs on the tape. As for TMs, such operations are of the following four types: (a) reading the symbol in the scanned cell, (b) printing a symbol onto the scanned cell, (c) moving to the left by one cell, and (d) moving to the right by one cell. However, only the operations of type (b), (c), and (d)–the *active operations*–are actually determined by the control box, since the *read* operation takes place automatically after each active operation and at the very beginning of each computation by a SBTM (see next).

Before a SBTM begins its computation with a string η as input, the symbols of η are placed (from left to right) in consecutive cells of the tape (the *input cells*), one symbol per cell, whereas the remaining cells of the tape are left "empty", i.e., they contain a special *blank* symbol; the head is positioned over (i.e., scans) the *initial cell*, namely, the cell immediately to the left of the input cell containing the first (i.e., leftmost) symbol of η. This is the *initial configuration* for a SBTM with input η. Then, starting from an initial configuration, SBTMs execute their computation steps at discrete instants of time, beginning at time $i = 0$. Each computation step, but the initial one, consists of the execution of an active operation, of one of the types (b), (c), or (d), immediately followed by a *read* operation from the cell being scanned by the head. In particular, for $i > 0$, the active operation O performed at $\boldsymbol{Step}_i$ is determined by the control box of the SBTM as a function of a suffix $\mathcal{S}$ of the sequence of the computation steps executed up to $\boldsymbol{Step}_{i-1}$ (included). In this case, we say that the SBTM has executed the *computation rule* (or *c-rule*) $\mathcal{S} \rightarrowtriangle O$. Concerning $\boldsymbol{Step}_0$, it is also convenient to regard it as consisting of the execution of an active operation followed by a *read* operation, as the remaining steps. Thus, we think of $\boldsymbol{Step}_0$ as consisting of the initial activation of the SBTM-control box–the zero operation–immediately followed by a read operation (of the blank symbol contained in the initial cell). After executing a computation step, a SBTM proceeds to the next step, and so on, until it possibly *halts*; this happens when none of the c-rules $\mathcal{S} \rightarrowtriangle O$ of the SBTM can be executed, for any suffix $\mathcal{S}$ of the sequence of computation steps executed up to then. Any finite sequence of consecutively executed computation steps constitutes a *(computation) trace* of the SBTM, whereas a *complete trace* is a trace starting at the initial computation $\boldsymbol{Step}_0$. The *computation history* of a SBTM $\mathfrak{V}$ is the whole sequence of configurations assumed by the tape of $\mathfrak{V}$ (namely, tape inscriptions along with head position) during the execution of the computation steps of $\mathfrak{V}$.

From the above description, it emerges that a SBTM is essentially a (finite) collection of c-rules. In particular, a *deterministic* SBTM is a *suffix-free* collection of c-rules, namely a collection of c-rules containing no two distinct c-rules $\mathcal{S}' \rightarrowtriangle O'$ and $\mathcal{S}'' \rightarrowtriangle O''$ such that $\mathcal{S}'$ is a suffix of $\mathcal{S}''$.[4] (Thus, intuitively, at any given computation step, the active operation O that the head of a deterministic SBTM can perform, if any, is uniquely determined by the previous computation steps.)

[4] In this paper we are interested only in deterministic SBTMs.

Formal definition. To formally define SBTMs, we need a convenient representation of c-rules and their components (i.e., traces and active operations). If we represent the zero operation with the symbol $\odot$, the left and right head motions with the symbols $\curvearrowleft$ and $\curvearrowright$, respectively, and, for $x \in \mathscr{S}$, we represent the operation of printing x onto the scanned cell with the very same symbol x, then every trace $\mathcal{T}$ can be handily represented as a string (in the alphabet $\mathscr{S} \cup \mathscr{A}$) as follows. First of all, we designate each computation step $\boldsymbol{S}$ by a 2-symbol string $\mathscr{o}x$, where $\mathscr{o} \in \mathscr{S} \cup \mathscr{A}$ is the symbol denoting the active operation O performed by $\boldsymbol{S}$ (as explained earlier) and x is the symbol subsequently read from the tape by step $\boldsymbol{S}$, immediately after the execution of O. Then a trace $\mathcal{T}$ can be represented with the string resulting from concatenating the 2-symbol strings representing the computation steps in $\mathcal{T}$, in the same order in which they occur in it. For instance, let $\mathcal{T}$ consist of the two computation steps $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, where: (i) $\boldsymbol{S}_1$ consists of moving the head one cell to the right and then reading the symbol $\Box$ from the tape; and (ii) $\boldsymbol{S}_2$ consists of printing s_1 onto the scanned cell and then reading the same symbol s_1. Then, $\mathcal{T}$ is represented by the string $\curvearrowright\Box\mathsf{s}_1\mathsf{s}_1$. Observe also that any complete trace of a SBTM, i.e., a trace starting at the initial $\boldsymbol{Step}_0$, is represented by a string with prefix $\odot\Box$. Finally, we represent a c-rule $\mathcal{S} \twoheadrightarrow O$ as the ordered pair $(\sigma, \mathscr{o})$, where σ is the string representing the trace $\mathcal{S}$ (as described above), and $\mathscr{o}$ is the symbol denoting the active operation O.

At this point, a SBTM could be formally defined simply as the set of the ordered pairs $(\sigma, \mathscr{o})$ representing its c-rules $\mathcal{S} \twoheadrightarrow O$. However, such an approach would require particular care to avoid circularity, since the notion of traces of SBTMs is defined in terms of the very same notion of c-rules which we want to define. For the sake of minimality, we shall circumvent this circularity problem by simply admitting, among the ordered pairs representing 'genuine' c-rules, even those pairs $(\sigma, \mathscr{o})$ in which the string σ could possibly represent no valid trace. Thus we give the following definitions.

Definition 1. *A* c-rule *is an ordered pair $(\sigma, \mathscr{o})$, also written as $\sigma \twoheadrightarrow \mathscr{o}$, where σ is any string in the alphabet $\mathscr{S} \cup \mathscr{A}$ and $\mathscr{o} \in \mathscr{S} \cup \{\curvearrowleft, \curvearrowright\}$. A set of c-rules is* suffix-free*, if it contains no two distinct c-rules $\sigma' \twoheadrightarrow \mathscr{o}'$ and $\sigma'' \twoheadrightarrow \mathscr{o}''$ such that $\sigma' \sqsupseteq \sigma''$.*

Definition 2. *A* (deterministic) SBTM *is any finite, suffix-free set of c-rules. The* alphabet of a SBTM $\mathfrak{V}$ *is the set $\mathscr{S}_{\mathfrak{V}}$ of all the tape symbols, but the blank $\Box$, occurring in any of its c-rules.*

2.1 Formal semantics of SBTMs

Let $\mathfrak{V}$ be a SBTM. Suppose that $\mathfrak{V}$ is ran with some given input string $\eta \in (\mathscr{S}_{\mathfrak{V}})^*$. For any time instant $i \geqslant 0$, we denote with $\mathcal{T}_i^{\mathfrak{V}}(\eta)$ the (string representing the) complete trace of $\mathfrak{V}$ from $\boldsymbol{Step}_0$ up to $\boldsymbol{Step}_i$, and with $\mathcal{C}_i^{\mathfrak{V}}(\eta)$ the tape configuration reached at the end of $\boldsymbol{Step}_i$. Letting C be the cell scanned by the head at the end of $\boldsymbol{Step}_i$, we represent $\mathcal{C}_i^{\mathfrak{V}}(\eta)$ as the triple (λ, s, ρ) in which:

(a) λ is the string consisting of the symbols contained, at the end of $\boldsymbol{Step}_i$, in the (possibly empty) portion of the tape to the left of C from the leftmost cell that has been scanned in any of the computation steps up to $\boldsymbol{Step}_i$; (b) s is the symbol contained in C at the end of $\boldsymbol{Step}_i$; and (c) ρ is the string consisting of the symbols contained, at the end of $\boldsymbol{Step}_i$, in the (possibly empty) portion of the tape to the right of C up to the rightmost cell that either has been scanned in any of the computation steps up to $\boldsymbol{Step}_i$ or is an input cell.

We say that a trace $\mathcal{T}_i^{\mathfrak{V}}(\eta)$ *is terminal (relative to* $\mathfrak{V}$*)*, if there is no c-rule $\sigma \twoheadrightarrow \phi$ in $\mathfrak{V}$ such that σ is a suffix of $\mathcal{T}_i^{\mathfrak{V}}(\eta)$.

The formal definitions of $\mathcal{C}_i^{\mathfrak{V}}(\eta)$ and $\mathcal{T}_i^{\mathfrak{V}}(\eta)$, for $i \geqslant 0$, are provided recursively as follows. Initially, for $i = 0$, we put

$$\mathcal{C}_0^{\mathfrak{V}}(\eta) := (\varepsilon, \Box, \eta) \quad \text{and} \quad \mathcal{T}_0^{\mathfrak{V}}(\eta) := \odot\Box\,.$$

For $i > 0$, let $\mathcal{C}_{i-1}^{\mathfrak{V}}(\eta) := (\lambda, s, \rho)$. Then, if $\mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)$ is terminal, we put $\mathcal{C}_i^{\mathfrak{V}}(\eta) := \mathcal{C}_{i-1}^{\mathfrak{V}}(\eta)$ and $\mathcal{T}_i^{\mathfrak{V}}(\eta) := \mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)$. Otherwise, let $\sigma \twoheadrightarrow \phi$ be *the* c-rule of $\mathfrak{V}$ such that σ is a suffix of $\mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)$.[5] Then we put recursively:

$$\mathcal{C}_i^{\mathfrak{V}}(\eta) \;/\; \mathcal{T}_i^{\mathfrak{V}}(\eta) := \begin{cases} (\lambda, \phi, \rho) \;/\; \mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)\phi\phi, & \text{if } \phi \in \mathscr{S} \\ (\mathsf{Init}(\lambda), \mathsf{Last}(\lambda), s\,.\,\rho) \;/\; \mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)\curvearrowleft\mathsf{Last}(\lambda), & \text{if } \phi = \curvearrowleft \\ (\lambda\,.\,s, \mathsf{Head}(\rho), \mathsf{Tail}(\rho)) \;/\; \mathcal{T}_{i-1}^{\mathfrak{V}}(\eta)\curvearrowright\mathsf{Head}(\rho), & \text{if } \phi = \curvearrowright\,. \end{cases}$$

The sequence $\mathcal{C}_0^{\mathfrak{V}}(\eta), \mathcal{C}_1^{\mathfrak{V}}(\eta), \mathcal{C}_2^{\mathfrak{V}}(\eta), \ldots$ is the *computation history of* $\mathfrak{V}$ *with input* η. We say that $\mathfrak{V}$ *with input* η *halts, and produce as output a string* $\omega \in (\mathscr{S}_{\mathfrak{V}})^*$ (and write $\mathfrak{V}(\eta) \downarrow \omega$), if, for some $i \geqslant 0$, we have that $\mathcal{T}_i^{\mathfrak{V}}(\eta)$ is terminal and ω is the string obtained from $\lambda s \rho$ by deleting all occurrences of the symbol $\Box$, where $(\lambda, s, \rho) := \mathcal{C}_i^{\mathfrak{V}}(\eta)$.

As in the case of TMs, SBTMs can be used in three different modalities: (a) to compute *partial functions*, (b) as *string generators*, and (c) as *language acceptors*. Specifically, given a SBTM $\mathfrak{V}$, we say that:

(a) $\mathfrak{V}$ *computes a (partial) string function* f over $(\mathscr{S}_{\mathfrak{V}})^*$, if, for all $\eta, \omega \in (\mathscr{S}_{\mathfrak{V}})^*$,

$$\mathfrak{V}(\eta) \downarrow \omega \quad \text{iff} \quad \omega = f(\eta)\,.$$

(b) $\mathfrak{V}$ *generates a string* $\omega \in (\mathscr{S}_{\mathfrak{V}})^*$, if $\mathfrak{V}(\varepsilon) \downarrow \omega$.

Finally, in order to use SBTMs as language acceptors, we must first define what we mean for a SBTM to *accept/reject* its input. Since SBTMs have no control states, a possibility could be the following one. We extend the definition of a SBTM by including two new distinguished symbols, say the symbols $\mathbb{y}$ (for "Yes") and $\mathbb{n}$ (for "No"), such that for no c-rule $\sigma \twoheadrightarrow \phi$ in the SBTM it is the case that $\mathbb{y}$ or $\mathbb{n}$ occurs in σ (but possibly we can have that $\phi = \mathbb{y}$ or $\phi = \mathbb{n}$).[6] Then we say that

[5] Observe that, according to Definition 2 there is in fact exactly one such c-rule $\sigma \twoheadrightarrow \phi$.
[6] Thus, if the SBTM prints $\mathbb{y}$ or $\mathbb{n}$ then it halts.

(c) a SBTM $\mathfrak{V}$ *accepts* (resp., *rejects*) *an input string* $\eta \in (\mathscr{S}_{\mathfrak{V}} \setminus \{\mathtt{y}, \mathtt{n}\})^*$, if there is a time instant $i \geqslant 0$ such that $\mathsf{Last}(\mathcal{T}_i^{\mathfrak{V}}(\eta)) = \mathtt{y}$ (resp., $\mathsf{Last}(\mathcal{T}_i^{\mathfrak{V}}(\eta)) = \mathtt{n}$).

Example 1. Let us consider the SBTM $\mathfrak{V}$, consisting of the following c-rules

(1) $\odot\Box \rightarrow \curvearrowright$ (2) $\odot\Box\curvearrowright\Box \rightarrow \mathtt{n}$ (3) $\odot\Box\curvearrowright s \rightarrow s$ (4) $ss\curvearrowright t \rightarrow s$
(5) $\curvearrowright stt \rightarrow \curvearrowright$ (6) $\curvearrowright uss\curvearrowright\Box \rightarrow \mathtt{y}$ (7) $\curvearrowright sss\curvearrowright\Box \rightarrow \mathtt{n}$,

where $s, t, u \in \{\mathtt{s}_1, \mathtt{s}_2\}$, with $s \neq u$. Then $\mathfrak{V}$ accepts exactly the nonempty strings in $\{\mathtt{s}_1, \mathtt{s}_2\}^*$ whose leftmost and rightmost symbols are different, while rejecting all remaining strings in $\{\mathtt{s}_1, \mathtt{s}_2\}^*$. The SBTM $\mathfrak{V}$ behaves as follows. Starting with an input string η on its tape, $\mathfrak{V}$ initially moves one cell to the right (cf. c-rule (1)) and checks whether the newly scanned cell C is empty; if C is empty (which means that η is the empty string), then $\mathfrak{V}$ prints the symbol $\mathtt{n}$ onto C (cf. c-rule (2)) and halts, thus rejecting η; otherwise, if C contains a symbol $s \in \{\mathtt{s}_1, \mathtt{s}_2\}$, then $\mathfrak{V}$ prints back s onto C (cf. c-rule (3)). Then, each time $\mathfrak{V}$ scans a nonempty cell D, i.e., a cell containing a symbol $t \in \{\mathtt{s}_1, \mathtt{s}_2\}$, it checks which of the following two conditions holds, namely: (i) the cell D has just been reached after a right head motion preceded by a printing action of a symbol $s \in \{\mathtt{s}_1, \mathtt{s}_2\}$; (ii) the cell D has just been involved in a printing action preceded by a right head motion. In case (i), $\mathfrak{V}$ prints the symbol s over D (cf. c-rule (4)), otherwise, in case (ii), $\mathfrak{V}$ moves one cell to the right of D (cf. c-rule (5)). Finally, when an empty cell E is encountered (following a right head motion), it is checked whether the two symbols s and u previously read from the two adjacent cells to the left of E are different. If this is the case, $\mathfrak{V}$ prints the symbol $\mathtt{y}$ onto E (cf. c-rule (6)) and halts, thus accepting η; otherwise, if s and u are equal, $\mathfrak{V}$ prints the symbol $\mathtt{n}$ (cf. c-rule (7)) and halts, thus rejecting η. Observe that, during the computation, the symbols of the input string η, but the leftmost one, are in turn replaced by the leftmost symbol of η; hence, at any step, the leftmost symbol of η is remembered by the last printed symbol. □

3 Turing completeness of SBTMs

We prove the Turing completeness of SBTMs by constructing for every TM $\mathfrak{M}$ a SBTM $\langle\mathfrak{M}\rangle$ such that $\mathfrak{M}$ and $\langle\mathfrak{M}\rangle$ are equivalent in the following strong sense, namely, for each input string η: (a) $\mathfrak{M}$ halts iff $\langle\mathfrak{M}\rangle$ halts; and (b) when $\mathfrak{M}$ and $\langle\mathfrak{M}\rangle$ halt, they do with the same tape configuration, i.e., with the same tape content and the same head position.[7] (In fact, we shall see informally that the computations of $\mathfrak{M}$ and $\langle\mathfrak{M}\rangle$ are synchronized in an even stronger way.)

To begin with, let us review some useful notations and concepts pertaining to TMs. We assume that the control states that any TM $\mathfrak{M}$ can assume belong to the set $\mathscr{Q} = \{\mathtt{q}_0, \mathtt{q}_1, \mathtt{q}_2, \ldots\}$, where $\mathtt{q}_0$ is bound to denote the *initial state* of $\mathfrak{M}$. Turing machine's instructions are represented as quadruples of the form $(q, x, \mathscr{o}, p)$, where

[7] In Section 3.1 we will see, as well, how to construct, for each SBTM $\mathfrak{V}$, an equivalent TM $\langle\!|\mathfrak{V}|\!\rangle$ which simulates $\mathfrak{V}$.

$q, p \in \mathscr{Q}$, $x \in \mathscr{S}$, and $\mathscr{o} \in \mathscr{S} \cup \{\curvearrowleft, \curvearrowright\}$, whose meaning is that when a TM is in state q while reading the symbol x, the head performs the active operation represented by $\mathscr{o}$, and then the TM enters state p (see [2]). A (deterministic) TM is then formally defined as a finite set of quadruples of the above type, no two of which begin with the same state-symbol pair (q, x), and such that at least one quadruple begins with q_0. For every TM $\mathfrak{M} := \{(q_i, x_i, \mathscr{o}_i, p_i) : 0 \leqslant i \leqslant n\}$, where $n \geqslant 0$, we put:

$$\mathscr{Q}_{\mathfrak{M}} := \{q_i, p_i : 0 \leqslant i \leqslant n\} \quad \text{and} \quad \mathscr{S}_{\mathfrak{M}} := \{\square\} \cup (\{x_i, \mathscr{o}_i : 0 \leqslant i \leqslant n\} \setminus \{\curvearrowleft, \curvearrowright\}) .$$

Thus, $\mathscr{Q}_{\mathfrak{M}}$ and $\mathscr{S}_{\mathfrak{M}}$ are the *set of states* and the *tape alphabet* of $\mathfrak{M}$, respectively. We say that a TM $\mathfrak{M}$ *activates a state-symbol pair* $(q, x) \in \mathscr{Q}_{\mathfrak{M}} \times \mathscr{S}_{\mathfrak{M}}$ when $\mathfrak{M}$ reads the symbol x from the tape while in state q.[8]

Let $\mathfrak{M}$ be a given TM and $\mathscr{S}_{\mathfrak{M}}$ its tape alphabet. Also, let m be the smallest *positive* index such that $\mathscr{S}_{\mathfrak{M}} \subseteq \{\mathsf{s}_0, \mathsf{s}_1, \ldots, \mathsf{s}_m\}$. For simplicity, we shall write s_m as s (hence, $\mathsf{s} \neq \square$). In addition, for $x \in \mathscr{S}$, let the 2-symbol string xx be denoted by $(\!|x|\!)$.[9] The SBTM $\langle\mathfrak{M}\rangle$ intended to simulate the TM $\mathfrak{M}$ will encode each state q_i of $\mathfrak{M}$ with the trace $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)$, consisting of a sequence of $(i+4)$ printing steps which uniquely characterizes q_i. The simulation proceeds in such a way that when $\mathfrak{M}$ activates the state-symbol pair (q_i, x), (i) the string $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)$ turns out to be a suffix of the current complete trace of $\langle\mathfrak{M}\rangle$ and, additionally, (ii) $\mathfrak{M}$ and $\langle\mathfrak{M}\rangle$ have the same tape configuration.

The first state-symbol pair activated by $\mathfrak{M}$ is $(\mathsf{q}_0, \square)$. Thus we put into $\langle\mathfrak{M}\rangle$ the following block $\mathfrak{S}_0$ of c-rules:

$$\odot\square \rightarrowtriangle \mathsf{s},\ \odot\square(\!|\mathsf{s}|\!) \rightarrowtriangle \square,\ \odot\square(\!|\mathsf{s}|\!)(\!|\square|\!) \rightarrowtriangle \square,\ \odot\square(\!|\mathsf{s}|\!)(\!|\square|\!)^2 \rightarrowtriangle \mathsf{s},\ \odot\square(\!|\mathsf{s}|\!)(\!|\square|\!)^2(\!|\mathsf{s}|\!) \rightarrowtriangle \square .$$

The c-rules in $\mathfrak{S}_0$ have the effect to generate the complete trace $\odot\square(\!|\mathsf{s}|\!)(\!|\square|\!)^2(\!|\mathsf{s}|\!)(\!|\square|\!)$ (while leaving the tape as in its initial configuration), whose suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^2(\!|\mathsf{s}|\!)(\!|\square|\!)$ correctly encodes the activation of the pair $(\mathsf{q}_0, \square)$. Notice that the block $\mathfrak{S}_0$ is independent of the specific TM $\mathfrak{M}$.

Next, for each instruction $I := (\mathsf{q}_i, x, \mathscr{o}, \mathsf{q}_j)$ in $\mathfrak{M}$ we define a corresponding simulating block $\langle I\rangle$ of c-rules for $\langle\mathfrak{M}\rangle$. We distinguish the following cases:

Case $\mathscr{o} = y$, with $y \in \mathscr{S}$: In this case $\langle I\rangle$ consists of the following c-rules:

$$(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!) \rightarrowtriangle \mathsf{s}\,, \qquad (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)(\!|\mathsf{s}|\!)(\!|\square|\!)^k \rightarrowtriangle \square\,, \text{ for } 0 \leqslant k \leqslant j+1$$
$$(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2} \rightarrowtriangle \mathsf{s}\,, \qquad (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!) \rightarrowtriangle y\,.$$

[*Comment*: Assuming recursively that the current complete trace $\mathcal{T}$ of the simulating computation of $\langle\mathfrak{M}\rangle$ has the suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)$ (corresponding to the

[8] Note that, since a TM initially scans the blank preceding the leftmost symbol of its input string (see [2]), the pair $(\mathsf{q}_0, \square)$ is always activated by every TM at the very beginning of each of its computations.

[9] Thus, $(\!|x|\!)$ represents the computation step consisting of printing the symbol x and then reading the same symbol x (just printed) from the tape.

activation of the pair (q_i, x)), on a tape configuration $\mathcal{C}$ in which the head scans a cell C with the symbol x, the above block of c-rules has the effect of: (i) appending the new suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!)(\!|y|\!)$ to $\mathcal{T}$; and (ii) returning a tape configuration $\mathcal{C}'$ in which the head scans the cell C, now containing the symbol y, and that otherwise is identical to $\mathcal{C}$. Notice that the suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!)(\!|y|\!)$ encodes the pair state-symbol (q_j, y), which is the next pair to be activated by $\mathfrak{M}$.]

Case $\mathscr{o} \in \{\curvearrowleft, \curvearrowright\}$: In this case $\langle I \rangle$ consists of the c-rule $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!) \twoheadrightarrow \mathscr{o}$, plus the blocks $\langle I \rangle_z$, for each $z \in \mathscr{S}_{\mathfrak{M}}$, consisting of the following c-rules:

$$(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)\mathscr{o}z \twoheadrightarrow \mathsf{s}\,, \qquad (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)\mathscr{o}z(\!|\mathsf{s}|\!)(\!|\square|\!)^{k} \twoheadrightarrow \square\,, \text{ for } 0 \leqslant k \leqslant j+1$$

$$(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)\mathscr{o}z(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2} \twoheadrightarrow \mathsf{s}\,, \qquad (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)\mathscr{o}z(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!) \twoheadrightarrow z\,.$$

[*Comment*: For $z \in \mathscr{S}_{\mathfrak{M}}$, let us put $\langle I \rangle_z^+ := \langle I \rangle_z \cup \{(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!) \twoheadrightarrow \mathscr{o}\}$. For simplicity, let us suppose that $\mathscr{o} = \curvearrowleft$ (we can reason similarly in the case in which $\mathscr{o} = \curvearrowright$). Assuming recursively that the current complete trace $\mathcal{T}$ of the simulating computation of $\langle \mathfrak{M} \rangle$ has the suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)$ (corresponding to the activation of the pair (q_i, x)), on a tape configuration $\mathcal{C}$ in which the head scans a cell C and the cell L on the left of C contains the tape symbol $z \in \mathscr{S}_{\mathfrak{M}}$, the subblock $\langle I \rangle_z^+$ of $\langle I \rangle$ has the effect of: (i) appending to $\mathcal{T}$ the new suffix $\mathscr{o}z(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!)(\!|z|\!)$; and (ii) returning a new tape configuration $\mathcal{C}'$ in which the head scans the cell L and that otherwise is identical to $\mathcal{C}$. Notice that the suffix $(\!|\mathsf{s}|\!)(\!|\square|\!)^{j+2}(\!|\mathsf{s}|\!)(\!|z|\!)$ of the prolongated trace of $\langle \mathfrak{M} \rangle$ encodes the pair state-symbol (q_j, z), which is the next pair to be activated by $\mathfrak{M}$.]

Finally, we put:

$$\langle \mathfrak{M} \rangle := \mathfrak{S}_0 \cup \bigcup_{I \in \mathfrak{M}} \langle I \rangle\,,$$

completing the formal definition of $\langle \mathfrak{M} \rangle$.

It can easily be verified that the set of c-rules $\langle \mathfrak{M} \rangle$ just given is suffix-tree (and, therefore, correctly defines a SBTM according to Definition 2). Indeed, observe that, for each c-rule $\sigma \twoheadrightarrow \mathscr{o}$ in the set $\bigcup_{I \in \mathfrak{M}} \langle I \rangle$, the string σ starts with a prefix π of the form $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)$, for some $i \geqslant 0$ and $x \in \mathscr{S}_{\mathfrak{M}}$, such that, for each c-rule $\sigma' \twoheadrightarrow \mathscr{o}'$ in $\bigcup_{I \in \mathfrak{M}} \langle I \rangle$: (i) π is not a factor of $\mathsf{Tail}(\sigma')$; and (ii) if $\pi \sqsubseteq \sigma'$ and $|\sigma| = |\sigma'|$, then $\sigma = \sigma'$ and $\mathscr{o} = \mathscr{o}'$. Also, observe that each c-rule $\sigma \twoheadrightarrow \mathscr{o}$ in $\mathfrak{S}_0$ is such that $\mathsf{Head}(\sigma) = \odot$, and that $\odot$ does not occur in any c-rule in $\bigcup_{I \in \mathfrak{M}} \langle I \rangle$. Notice also that, apart from the trivial case in which $\mathscr{S}_{\mathfrak{M}} = \{\square\}$ (in which case we have $\mathsf{s} = \mathsf{s}_1 \notin \mathscr{S}_{\mathfrak{M}}$), the c-rules of $\langle \mathfrak{M} \rangle$ use only tape symbols in the alphabet $\mathscr{S}_{\mathfrak{M}}$ of $\mathfrak{M}$. Finally, we observe that the above comments to the definitions of the blocks $\langle I \rangle$ of c-rules, for each instruction $I \in \mathfrak{M}$, could be easily translated into a formal proof of the fact that $\langle \mathfrak{M} \rangle$ correctly simulates $\mathfrak{M}$ in the strong sense described at the beginning of the section.

Complexity of various encodings. As discussed above, the main idea behind the construction of $\langle \mathfrak{M} \rangle$ is to simulate the activation by $\mathfrak{M}$ of each state-symbol pair (q_i, x) by means of the trace $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)(\!|x|\!)$ of $\langle \mathfrak{M} \rangle$, where $(\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)$

encodes the state q_i. This approach can be generalized as follows. First of all, we associate to each state q_i of $\mathfrak{M}$ a suitable *codeword* $\mathsf{c}(\mathsf{q}_i)$ in the alphabet $\{(\!|\mathsf{s}|\!), (\!|\square|\!)\}$ (in which each of the two strings $(\!|\mathsf{s}|\!)$ and $(\!|\square|\!)$ is temporarily regarded as a single symbol) and then simulate the activation of any pair (q_i, x) by means of the trace $\mathsf{c}(\mathsf{q}_i)\,.\,(\!|x|\!)$. The case $\mathsf{c}(\mathsf{q}_i) := (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+2}(\!|\mathsf{s}|\!)$ corresponds to the approach adopted in the proof sketched above. For each choice of the particular encoding c of the states of $\mathfrak{M}$, the definitions of the sets $\mathfrak{S}_0$ and $\langle I\rangle$, for $I \in \mathfrak{M}$, provided before, generalize straightforwardly, thus obtaining the corresponding sets of c-rules $\mathfrak{S}_0^{\mathsf{c}}$ and $\langle I\rangle^{\mathsf{c}}$, for $I \in \mathfrak{M}$ (details are omitted for brevity). Clearly, we are interested only in encodings c which are *admissible*, in the sense that the resulting set of c-rules

$$\langle \mathfrak{M}\rangle^{\mathsf{c}} := \mathfrak{S}_0^{\mathsf{c}} \cup \bigcup_{I \in \mathfrak{M}} \langle I\rangle^{\mathsf{c}}$$

is suffix-free. For instance, a family of admissible encodings is provided by the functions c_h, for $h \geqslant 2$, such that $\mathsf{c}_h(\mathsf{q}_i) := (\!|\mathsf{s}|\!)(\!|\square|\!)^{i+h}(\!|\mathsf{s}|\!)$. Again, for $h = 2$ we obtain the encoding c_2 adopted in our previous proof. If we assume that $\mathscr{Q}_{\mathfrak{M}} = \{\mathsf{q}_0, \mathsf{q}_1, \ldots, \mathsf{q}_{k-1}\}$, for some $k \geqslant 1$ (in which case we say that $\mathfrak{M}$ is *tight*), each of the encodings c_h generates a SBTM $\langle \mathfrak{M}\rangle^{\mathsf{c}_h}$ whose longest c-rule has linear length in the number k of the states of $\mathfrak{M}$. More in general, for each TM $\mathfrak{T}$ and SBTM $\mathfrak{V}$, let us define the *sizes* $\|\mathfrak{T}\|$ and $\|\mathfrak{V}\|$ by putting:

$$\|\mathfrak{T}\| := |\mathscr{Q}_{\mathfrak{T}}| \quad \text{and} \quad \|\mathfrak{V}\| := \max\{|\sigma| : \sigma \twoheadrightarrow \mathscr{o} \in \mathfrak{V}, \text{ for some } \mathscr{o} \in \mathscr{S} \cup \{\curvearrowleft, \curvearrowright\}\}$$

(where $|\mathscr{Q}_{\mathfrak{T}}|$ denotes the cardinality of $\mathscr{Q}_{\mathfrak{T}}$). Then, using the asymptotic notation Θ, we have that $\|\langle \mathfrak{M}\rangle^{\mathsf{c}_h}\| = \Theta(\|\mathfrak{M}\|)$, for each $h \geqslant 2$. This indeed follows from the facts that (**A**) for each encoding c, $\|\langle \mathfrak{M}\rangle^{\mathsf{c}}\| = \Theta(\max(\{|\mathsf{c}(q)| : q \in \mathscr{Q}_{\mathfrak{M}}\}))$, and (**B**) $|\mathsf{c}_h(\mathsf{q}_i)| = 2\cdot(2+i+h)$, for $h \geqslant 2$. Notice, however, that more "compact" admissible encodings c can be devised such that $\|\langle \mathfrak{M}\rangle^{\mathsf{c}}\| = \Theta(\log \|\mathfrak{M}\|)$. Indeed, let $\zeta_0, \zeta_1, \zeta_2, \ldots$ be the list in quasi-lexicographic order[10] of the nonempty strings ζ in the alphabet $\{(\!|\mathsf{s}|\!), (\!|\square|\!)\}$ such that ζ contains no two consecutive occurrences of $(\!|\mathsf{s}|\!)$, and consider the encoding f such that $\mathsf{f}(\mathsf{q}_i) := (\!|\mathsf{s}|\!)^2(\!|\square|\!)\zeta_i(\!|\square|\!)(\!|\mathsf{s}|\!)^2$, for each $i \geqslant 0$. By reasoning much as we did for $\langle \mathfrak{M}\rangle$, it can be shown that $\langle \mathfrak{M}\rangle^{\mathsf{f}}$ is suffix-free, i.e., f is admissible.[11] Concerning the size of $\langle \mathfrak{M}\rangle^{\mathsf{f}}$, we show next that it is logarithmic in the size of $\mathfrak{M}$, provided that $\mathfrak{M}$ is tight. To begin with, it can easily be verified that, for each $n \geqslant 0$, the number of strings of length n in the alphabet $\{(\!|\mathsf{s}|\!), (\!|\square|\!)\}$ containing no two consecutive $(\!|\mathsf{s}|\!)$'s is exactly F_{n+2}, where F_n is the nth *Fibonacci number*.[12] Exploiting the identity $\sum_{j=1}^{n} F_j = F_{n+2} - 1$

[10] Hence, the strings ζ_i's are firstly ordered by their length and then lexicographically, where w.l.o.g. we conventionally assume that $(\!|\square|\!) < (\!|\mathsf{s}|\!)$.

[11] Note that, if we remove from $\mathsf{f}(\mathsf{q}_i)$ the "symbol" $(\!|\square|\!)$ surrounding ζ_i, the resulting encoding $\mathsf{g}(\mathsf{q}_i) := (\!|\mathsf{s}|\!)^2\zeta_i(\!|\mathsf{s}|\!)^2$ turns out to be not admissible. For, consider the case in which the TM $\mathfrak{M}$ contains the instruction $I := (\mathsf{q}_0, \mathsf{s}, x, \mathsf{q}_3)$, where $x \in \mathscr{S}$. Then, since $\zeta_0 = (\!|\square|\!)$ and $\zeta_3 = (\!|\square|\!)(\!|\mathsf{s}|\!)$, we have $\mathsf{g}(\mathsf{q}_0) = (\!|\mathsf{s}|\!)^2(\!|\square|\!)(\!|\mathsf{s}|\!)^2$ and $\mathsf{g}(\mathsf{q}_3) = (\!|\mathsf{s}|\!)^2(\!|\square|\!)(\!|\mathsf{s}|\!)^3$, so that the set $\langle I\rangle^{\mathsf{g}}$ contains the c-rules $(\!|\mathsf{s}|\!)^2(\!|\square|\!)(\!|\mathsf{s}|\!)^3 \twoheadrightarrow \mathsf{s}$ and $(\!|\mathsf{s}|\!)^2(\!|\square|\!)(\!|\mathsf{s}|\!)^5(\!|\square|\!)(\!|\mathsf{s}|\!)^3 \twoheadrightarrow x$, plainly implying that $\langle \mathfrak{M}\rangle^{\mathsf{g}}$ is not suffix-free.

[12] Thus, $F_0 = 0$, $F_1 = 1$, and $F_{k+2} = F_k + F_{k+1}$, for $k \geqslant 0$.

(for $n \geqslant 0$), we have

$$F_{|\zeta_i|+3} - 3 = \sum_{j=1}^{|\zeta_i|-1} F_{j+2} \leqslant i < \sum_{j=1}^{|\zeta_i|} F_{j+2} = F_{|\zeta_i|+4} - 3$$

for each $i \geqslant 2$. Given that $F_n = \Theta(\phi^n)$ (in fact $\frac{F_n}{\phi^n} \to \frac{1}{\sqrt{5}}$), where $\phi := \frac{1+\sqrt{5}}{2} \approx 1.618$ is the *golden ratio*, the latter inequalities imply $|\zeta_i| = \Theta(\log_\phi i)$. Hence we have $|\mathsf{f}(\mathsf{q}_i)| = \Theta(\log_\phi i)$, which yields, by (**A**) above and by the tightness of $\mathfrak{M}$,

$$\|\langle\mathfrak{M}\rangle^{\mathsf{f}}\| = \Theta(\log_\phi \|\mathfrak{M}\|)\,,$$

proving that the size of $\langle\mathfrak{M}\rangle^{\mathsf{f}}$ is logarithmic in the size of $\mathfrak{M}$, when $\mathfrak{M}$ is tight.

3.1 Standard TMs with bounded temporal memory

The intuitive considerations of Section 1 concerning the independence of the computational power of TMs from the property of these devices to remember subsequences of arbitrarily old actions within their control states can be made more precise by using a notion of *trace of* TMs identical to that of SBTM's trace introduced earlier. More precisely, a trace of a TM $\mathfrak{M}$ is a sequence of consecutive computation steps of $\mathfrak{M}$, where a computation step consists in the execution of an active operation of the scanning head on the tape (i.e., printing a symbol or a left/right motion) followed by the subsequent reading of the symbol contained in the newly scanned cell, as in the case of SBTMs.

Next, suppose that a TM $\mathfrak{M}$ performs an active operation O on its tape, during a given computation step $\boldsymbol{S}$. Then, letting $\mathcal{T}$ be the *complete trace* of $\mathfrak{M}$ consisting of the whole sequence of computation steps preceding $\boldsymbol{S}$, it can be readily verified that O is *uniquely determined* by $\mathcal{T}$; i.e., there is a function, $\Upsilon_{\mathfrak{M}}$, which maps each complete trace $\mathcal{T}$ to the particular active operation $O := \Upsilon_{\mathfrak{M}}(\mathcal{T})$ to be performed next. Now, let us define the class $\boldsymbol{\Omega}$ of *TMs with bounded temporal memory*. Specifically, the class $\boldsymbol{\Omega}$ consists of all TMs $\mathfrak{M}$ for which there is a constant $\ell_{\mathfrak{M}}$ (depending on $\mathfrak{M}$) such that the (active) operations to be performed next on the tape can be determined only by suffixes of the complete traces of $\mathfrak{M}$ consisting of at most $\ell_{\mathfrak{M}}$ consecutive computation steps; i.e., more formally, the function $\Upsilon_{\mathfrak{M}}$ is such that $\Upsilon_{\mathfrak{M}}(\mathcal{T}') = \Upsilon_{\mathfrak{M}}(\mathcal{T}'')$, for any two complete traces $\mathcal{T}'$ and $\mathcal{T}''$ which share a common suffix of $\ell_{\mathfrak{M}}$ computation steps. Intuitively, TMs in the class $\boldsymbol{\Omega}$ do not care of (or *forget*) actions strictly older than $\ell_{\mathfrak{M}}$ computation steps. Then, the question arises whether the computational power of TMs decreases when we restrict to the class $\boldsymbol{\Omega}$ above. As will be outlined below, given any SBTM $\mathfrak{V}$, an equivalent TM $\langle\!|\mathfrak{V}|\!\rangle$ can be constructed such that $\langle\!|\mathfrak{V}|\!\rangle \in \boldsymbol{\Omega}$; therefore, since SBTMs are Turing complete, it follows that every TM $\mathfrak{M}$ is equivalent to some TM $\mathfrak{T} \in \boldsymbol{\Omega}$ (e.g., $\mathfrak{T} := \langle\!|\langle\mathfrak{M}\rangle|\!\rangle$). Hence the answer to the above question is that the class $\boldsymbol{\Omega}$ is, in fact, Turing complete.

The basic idea of the construction of the TM $\langle\!|\mathfrak{V}|\!\rangle$, for a given SBTM $\mathfrak{V}$, is to use the control states of $\langle\!|\mathfrak{V}|\!\rangle$ to store in turn the strings θ of length $2\ell - 1$, with

$\ell := \frac{\|\mathfrak{V}\|}{2}$, such that $\theta\,.\,x$ represents the last ℓ computation steps of the current (complete) trace of $\langle\!|\mathfrak{V}|\!\rangle$, where x is the symbol currently scanned by the head of $\langle\!|\mathfrak{V}|\!\rangle$. When $\langle\!|\mathfrak{V}|\!\rangle$ assumes a state q, while reading the symbol x from the tape, the particular operation O that $\langle\!|\mathfrak{V}|\!\rangle$ has to perform next is determined by the c-rule $\sigma\,.\,x \rightarrowtriangle \mathscr{o}$ of $\mathfrak{V}$ such that σ is a suffix of the string θ stored within q, i.e, O is the operation represented by $\mathscr{o}$. Formally, the construction of $\langle\!|\mathfrak{V}|\!\rangle$ goes as follows. Let $\theta_0, \theta_1, \theta_2, \ldots$ be any listing of the strings in the alphabet $\mathscr{B} := \mathscr{S}_{\mathfrak{V}} \cup \{\square, \odot, \curvearrowleft, \curvearrowright\}$, where $\theta_0 = \odot$, and let $\mathbb{Q}\colon \mathscr{B}^* \longrightarrow \{\mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2, \ldots\}$ be the function such that, for each $i \geqslant 0$: if $|\theta_i| < 2\ell$, then $\mathbb{Q}(\theta_i) = \mathsf{q}_i$; otherwise, if $|\theta_i| \geqslant 2\ell$, then $\mathbb{Q}(\theta_i) = \mathbb{Q}(\overline{\theta_i})$, where $\overline{\theta_i}$ is the suffix of θ_i of length $2\ell - 1$.[13] Then, we let $\langle\!|\mathfrak{V}|\!\rangle$ be the Turing machine whose instructions are all the quadruples $(\mathbb{Q}(\theta), x, \mathscr{o}, \mathbb{Q}(\theta x \mathscr{o}))$ such that $\sigma\,.\,x \rightarrowtriangle \mathscr{o} \in \mathfrak{V}$ and $\sigma \sqsupseteq \theta$, where $\theta \in \mathscr{B}^*$, $x \in \mathscr{S}_{\mathfrak{V}} \cup \{\square\}$, and $\mathscr{o} \in \mathscr{S}_{\mathfrak{V}} \cup \{\square, \curvearrowleft, \curvearrowright\}$. It can be verified that $\langle\!|\mathfrak{V}|\!\rangle$ is indeed a TM equivalent to $\mathfrak{V}$ and that $\langle\!|\mathfrak{V}|\!\rangle \in \boldsymbol{\Omega}$ (details are omitted for brevity).

4 Conclusions

We have presented a variant of the Turing machine model of computation with no control states, named SBTM. In each computation step, the operation performed by the head is determined by the symbol currently scanned and by a suffix of bounded length of the sequence of the computation steps previously executed. We have shown that SBTMs are Turing complete, namely they are computationally as powerful as standard Turing machines. In addition, based on the Turing completeness of SBTMs, we have also shown that the computational power of TMs is independent of their ability to remember subsequences of arbitrarily old actions within their control states.

We plan to investigate further computational properties of SBTMs and, in particular, properties related to the following notion of string complexity which naturally arises in this context: given a string η, the SBTM-*complexity of* η is defined as the size of a minimum sized SBTM $\mathfrak{V}$ such that: (i) the alphabet of $\mathfrak{V}$ consists precisely of the symbols occurring in η; and (ii) $\mathfrak{V}$ generates η.

Acknowledgments

This work has been partially supported by the FIR project COMPACT: "Computazione affidabile su testi firmati" (code D84C46) and by project PRISMA PON04a2_A, funded by the Italian Ministry of University and Research within the PON 2007-2013 "Smart cities and communities" framework.

References

1. Joshua J. Arulanandham: Unconventional "Stateless" Turing-Like Machines. In: Selim G. Akl, Cristian S. Calude, Michael J. Dinneen, Grzegorz Rozenberg, and

[13] Note that $\mathbb{Q}(\odot) = \mathsf{q}_0$.

H. Todd Wareham, editors, Unconventional Computation, volume 4618 of *Lecture Notes in Computer Science*, pages 55–61. Springer Berlin Heidelberg, 2007.
2. Martin Davis, Ron Sigal, Elaine J. Weyuker: *Computability, complexity, and languages*. Second Edition, Academic Press, 1994.
3. M. Kutrib, H. Messerschmidt, O. Friedrich: On stateless deterministic restarting automata. *Acta Informatica*, volume 47, pages 391–412. Springer-Verlag, 2010.
4. O. H. Ibarra, J. Karhumäki, A. Okhotin: On Stateless Multihead Automata: Hierarchies and the Emptiness Problem. In: E. S. Laber, C. Bornstein, L. T. Nogueira, L. Faria, editors, *LATIN 2008: Theoretical Informatics*, volume 4957 of *Lecture Notes in Computer Science*, pages 94–105, Springer Berlin Heidelberg, 2008.
5. S. Muthukrishnan: Data Streams: Algorithms and Applications. Foundations and Trends in Theoretical Computer Science, volume 1, issue 2, pages 117–236, 2005.
6. Claude E. Shannon: A universal Turing machine with two internal states. Automata Studies, *Annals of Mathematics Studies*, volume 34, pages 157–165, 1956.
7. L. Valiant: Decision procedures for families of deterministic pushdown automata. PhD thesis, University of Warwick, 1973.

Interval Temporal Logic Model Checking Based on Track Bisimilarity and Prefix Sampling

Laura Bozzelli[1], Alberto Molinari[2], Angelo Montanari[2](✉), Adriano Peron[3], and Pietro Sala[4]

[1] Technical University of Madrid (UPM), Madrid, Spain
laura.bozzelli@fi.upm.es
[2] University of Udine, Udine, Italy
molinari.alberto@gmail.com, angelo.montanari@uniud.it
[3] University of Napoli "Federico II", Napoli, Italy
adrperon@unina.it
[4] University of Verona, Verona, Italy
pietro.sala@univr.it

Abstract. Since the late 80s, LTL and CTL model checking have been extensively applied in various areas of computer science and AI. Even though they proved themselves to be quite successful in many application domains, there are some relevant temporal conditions which are inherently "interval based" (this is the case, for instance, with telic statements like "the astronaut must walk home in an hour" and temporal aggregations like "the average speed of the rover cannot exceed the established threshold") and thus cannot be properly modelled by point-based temporal logics. In general, to check interval properties of the behavior of a system, one needs to collect information about states into behavior stretches, which amounts to interpreting each finite sequence of states as an interval and to suitably defining its labelling on the basis of the labelling of the states that compose it.

In order to deal with these properties, a model checking framework based on Halpern and Shoham's interval temporal logic (HS for short) and its fragments has been recently proposed and systematically investigated in the literature. In this paper, we give an original proof of **EXPSPACE** membership of the model checking problem for the HS fragment $\mathsf{A\overline{A}B\overline{B}E}$ (resp., $\mathsf{A\overline{A}E\overline{B}E}$) of Allen's interval relations *meets*, *met-by*, *started-by* (resp., *finished-by*), *starts*, and *finishes*. The proof exploits *track bisimilarity* and *prefix sampling*, and it turns out to be much simpler than the previously known one. In addition, it improves some upper bounds.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 49–61 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

1 Introduction

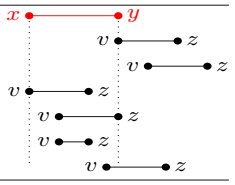
Interval temporal logics (ITLs) have been proposed as an alternative setting for reasoning about time [7, 17, 20] with respect to standard, point-based logics such as LTL [18] and CTL [6]. ITLs take intervals, rather than points, as their primitive entities, and their expressiveness enables them to specify, for instance, actions with duration, accomplishments, and temporal aggregations, which are inherently "interval-based" and cannot be expressed by point-based logics.

In this paper, we make use of ITLs as the specification language in model checking (MC), one of the most successful techniques in the area of formal methods, which allows a user to automatically check whether some desired properties of a system, specified by a temporal logic formula, hold over a model of it (usually a Kripke structure). In order to verify interval properties of computations, one needs to collect information about states into computation stretches: each finite path in a Kripke structure is interpreted as an interval, whose labelling is defined on the basis of the labelling of the component states. We focus our attention on *Halpern and Shoham's modal logic of time intervals* (HS) [7] which features one modality for each of the 13 possible ordering relations between pairs of intervals (the so-called Allen's relations [1]), apart from equality. Its *satisfiability problem* turns out to be undecidable for all relevant (classes of) linear orders [7]. The same holds for most fragments of HS [3, 8, 12]; however, some exceptions exist, e.g., the *logic of temporal neighbourhood* and the *logic of sub-intervals* [4, 5].

The *MC problem* for HS has been considered only very recently [2, 9, 10, 11, 13, 14, 15, 16]. In [13], Molinari et al. study MC for full HS (under the homogeneity assumption [19]). They introduce the problem and prove its non-elementary decidability. In [2], the authors prove its **EXPSPACE**-hardness. Since then, the attention was also brought to the fragments of HS, which, similarly to what happens with satisfiability, are often computationally better. The MC problem for *epistemic extensions* of some HS fragments has been investigated by Lomuscio and Michaliszyn [9, 10, 11] (a detailed account of their results can be found in [13]). However, their semantic assumptions differ from those of [13] (we make the same assumptions here), thus making it difficult to compare the two research lines.

In this paper, we study the MC problem for the HS fragment $\mathsf{A\overline{A}B\overline{B}E}$ (resp., $\mathsf{A\overline{A}E\overline{B}\overline{E}}$), whose modalities allow one to access intervals which are met by/meet the current one, or are prefixes (resp., suffixes) or right/left-extensions of it. In [15], the authors show that the problem is in **EXPSPACE**. The MC algorithm they describe exploits the possibility of finding, for each track of a Kripke structure, a satisfiability-preserving track of bounded length, called a *track representative*. Thus, the algorithm needs to check only tracks with a bounded maximum length. In [14], they prove the problem to be **PSPACE**-hard. The proof of membership to **EXPSPACE** is rather involved, and two very technical notions, namely, the notions of *scan function* and *configuration*, are introduced in order to determine the aforementioned bound to the length of representatives. Here, we provide a much easier proof, which leads to another class of track representatives, with the same purpose of those of [15], but shorter in general.

Table 1. Allen's relations and corresponding HS modalities.

Allen relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A \rangle$	$[x,y]\mathcal{R}_A[v,z] \iff y = v$	
BEFORE	$\langle L \rangle$	$[x,y]\mathcal{R}_L[v,z] \iff y < v$	
STARTED-BY	$\langle B \rangle$	$[x,y]\mathcal{R}_B[v,z] \iff x = v \wedge z < y$	
FINISHED-BY	$\langle E \rangle$	$[x,y]\mathcal{R}_E[v,z] \iff y = z \wedge x < v$	
CONTAINS	$\langle D \rangle$	$[x,y]\mathcal{R}_D[v,z] \iff x < v \wedge z < y$	
OVERLAPS	$\langle O \rangle$	$[x,y]\mathcal{R}_O[v,z] \iff x < v < y < z$	

The paper is organized as follows. In the next section, we introduce the fundamental elements of the MC problem for HS, and we give a short account of the known complexity results about MC for HS fragments. In Sect. 3, we introduce the notion of *bisimilarity* among tracks, that is exploited in Sect. 4, along with *prefix samplings*, to build, given a (generic) track ρ, a track ρ' of bounded length, and indistinguishable from ρ with respect to satisfiability of $\mathsf{A\overline{A}B\overline{B}E}$ formulas, having nesting depth of modality $\langle \mathrm{B} \rangle$ up to some $k \geq 0$.

2 Preliminaries

The interval temporal logic HS. An interval algebra to reason about intervals and their relative order was proposed by Allen in [1], while a systematic logical study of interval representation and reasoning was done a few years later by Halpern and Shoham, who introduced the interval temporal logic HS featuring one modality for each Allen relation, but equality [7]. Table 1 depicts 6 of the 13 Allen's relations, together with the corresponding HS (existential) modalities. The other 7 relations are the 6 inverse relations (given a binary relation $\mathcal{R}$, the inverse relation $\overline{\mathcal{R}}$ is such that $b\overline{\mathcal{R}}a$ if and only if $a\mathcal{R}b$) and equality.

The language of HS consists of a set of proposition letters $\mathcal{AP}$, the Boolean connectives $\neg$ and $\wedge$, and a temporal modality for each of the (non trivial) Allen's relations, i.e., $\langle \mathrm{A} \rangle$, $\langle \mathrm{L} \rangle$, $\langle \mathrm{B} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \mathrm{D} \rangle$, $\langle \mathrm{O} \rangle$, $\langle \overline{\mathrm{A}} \rangle$, $\langle \overline{\mathrm{L}} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, $\langle \overline{\mathrm{E}} \rangle$, $\langle \overline{\mathrm{D}} \rangle$, and $\langle \overline{\mathrm{O}} \rangle$. HS formulas are defined by the grammar $\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \langle X \rangle \psi \mid \langle \overline{X} \rangle \psi$, where $p \in \mathcal{AP}$ and $X \in \{A, L, B, E, D, O\}$. In the following, we will also exploit the other usual logical connectives (disjunction $\vee$, implication $\rightarrow$, and double implication $\leftrightarrow$) as abbreviations. Furthermore, for any modality X, the dual universal modalities $[X]\psi$ and $[\overline{X}]\psi$ are defined as $\neg\langle X \rangle \neg\psi$ and $\neg\langle \overline{X} \rangle \neg\psi$, respectively.

The joint nesting depth of B and E in a formula ψ, denoted by $\mathrm{d}_{\mathrm{BE}}(\psi)$, is defined as: (i) $\mathrm{d}_{\mathrm{BE}}(p) = 0$, for any $p \in \mathcal{AP}$; (ii) $\mathrm{d}_{\mathrm{BE}}(\neg\psi) = \mathrm{d}_{\mathrm{BE}}(\psi)$; (iii) $\mathrm{d}_{\mathrm{BE}}(\psi \wedge \phi) = \max\{\mathrm{d}_{\mathrm{BE}}(\psi), \mathrm{d}_{\mathrm{BE}}(\phi)\}$; (iv) $\mathrm{d}_{\mathrm{BE}}(\langle \mathrm{X} \rangle \psi) = 1 + \mathrm{d}_{\mathrm{BE}}(\psi)$, when $X = B$ or $X = E$; (v) $\mathrm{d}_{\mathrm{BE}}(\langle \mathrm{X} \rangle \psi) = \mathrm{d}_{\mathrm{BE}}(\psi)$, when both $X \neq B$ and $X \neq E$. If we consider formulas ψ of HS fragments devoid of E (resp., B), the nesting depth of modality B (resp., E) in ψ, denoted as $\mathrm{d}_{\mathrm{B}}(\psi)$ (resp., $\mathrm{d}_{\mathrm{E}}(\psi)$), accounts for modality B (resp., E) only, and $\mathrm{d}_{\mathrm{B}}(\psi) = \mathrm{d}_{\mathrm{BE}}(\psi)$ (resp., $\mathrm{d}_{\mathrm{E}}(\psi) = \mathrm{d}_{\mathrm{BE}}(\psi)$).

Given any subset of Allen's relations $\{X_1, .., X_n\}$, we denote by $\mathsf{X_1 \cdots X_n}$ the HS fragment featuring existential (and universal) modalities for $X_1, .., X_n$ only.

W.l.o.g., we assume the *non-strict semantics of HS*, which admits intervals consisting of a single point[5]. Under such an assumption, all HS modalities can be expressed in terms of modalities $\langle \mathrm{B} \rangle, \langle \mathrm{E} \rangle, \langle \overline{\mathrm{B}} \rangle$, and $\langle \overline{\mathrm{E}} \rangle$ [20]. HS can thus be regarded as a multi-modal logic with these 4 primitive modalities and its semantics can be defined over a multi-modal Kripke structure, called *abstract interval model*, where intervals are treated as atomic objects and Allen's relations as binary relations between pairs of intervals. Since later we will focus on the HS fragments $\mathsf{A\overline{A}E\overline{B}E}$ and $\mathsf{A\overline{A}B\overline{B}E}$—which do not feature $\langle \mathrm{B} \rangle$ and $\langle \mathrm{E} \rangle$ respectively—we add both $\langle \mathrm{A} \rangle$ and $\langle \overline{\mathrm{A}} \rangle$ to the considered set of HS modalities.

Definition 1. *[13] An* abstract interval model *is a tuple* $\mathcal{A}=(\mathcal{AP},\mathbb{I},A_{\mathbb{I}},B_{\mathbb{I}},E_{\mathbb{I}},\sigma)$*, where* $\mathcal{AP}$ *is a set of proposition letters,* $\mathbb{I}$ *is a possibly infinite set of atomic objects (worlds),* $A_{\mathbb{I}}$*,* $B_{\mathbb{I}}$*, and* $E_{\mathbb{I}}$ *are three binary relations over* $\mathbb{I}$*, and* $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$ *is a (total) labeling function, assigning a set of proposition letters to each world.*

In the interval setting, $\mathbb{I}$ is interpreted as a set of intervals and $A_{\mathbb{I}}$, $B_{\mathbb{I}}$, and $E_{\mathbb{I}}$ as Allen's relations A (*meets*), B (*started-by*), and E (*finished-by*), respectively; σ assigns to each interval in $\mathbb{I}$ the set of proposition letters that hold over it.

Given an abstract interval model $\mathcal{A} = (\mathcal{AP}, \mathbb{I}, A_{\mathbb{I}}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$ and an interval $I \in \mathbb{I}$, the truth of an HS formula over I is inductively defined as follows:

- $\mathcal{A}, I \models p$ iff $p \in \sigma(I)$, for any $p \in \mathcal{AP}$;
- $\mathcal{A}, I \models \neg\psi$ iff it is not true that $\mathcal{A}, I \models \psi$ (also denoted as $\mathcal{A}, I \not\models \psi$);
- $\mathcal{A}, I \models \psi \wedge \phi$ iff $\mathcal{A}, I \models \psi$ and $\mathcal{A}, I \models \phi$;
- $\mathcal{A}, I \models \langle X \rangle \psi$, for $X \in \{A, B, E\}$, iff there is $J \in \mathbb{I}$ s.t. $I \, X_{\mathbb{I}} \, J$ and $\mathcal{A}, J \models \psi$;
- $\mathcal{A}, I \models \langle \overline{X} \rangle \psi$, for $\overline{X} \in \{\overline{A}, \overline{B}, \overline{E}\}$, iff there is $J \in \mathbb{I}$ s.t. $J \, X_{\mathbb{I}} \, I$ and $\mathcal{A}, J \models \psi$.

Kripke structures and abstract interval models. In the context of MC, finite state systems are usually modelled as finite Kripke structures. In [13], the authors define a mapping from Kripke structures to abstract interval models, that allows one to specify interval properties of computations by means of HS formulas.

Definition 2. *A* finite Kripke structure *is a tuple* $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$*, where* $\mathcal{AP}$ *is a set of proposition letters,* W *is a finite set of states,* $\delta \subseteq W \times W$ *is a left-total relation between pairs of states,* $\mu : W \mapsto 2^{\mathcal{AP}}$ *is a total labelling function, and* $w_0 \in W$ *is the initial state.*

For all $w \in W$, $\mu(w)$ is the set of proposition letters that hold at w, while δ is the transition relation that describes the evolution of the system over time.

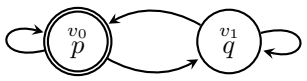


Fig. 1. The Kripke structure $\mathcal{K}_a$.

Fig. 1 depicts the finite Kripke structure $\mathcal{K}_a = (\{p, q\}, \{v_0, v_1\}, \delta, \mu, v_0)$, where $\delta = \{(v_0, v_0), (v_0, v_1), (v_1, v_0), (v_1, v_1)\}$, $\mu(v_0) = \{p\}$, and $\mu(v_1) = \{q\}$. The initial state v_0 is identified by a double circle.

Definition 3. *A* track ρ *of a finite Kripke structure* $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$ *is a finite sequence of states* $v_1 \cdots v_n$*, with* $n \geq 1$*, s.t.* $(v_i, v_{i+1}) \in \delta$ *for* $i \in [1, n-1]$.

[5] All the results we prove in the paper hold for the strict semantics as well.

Let $\mathrm{Trk}_{\mathcal{K}}$ be the (infinite) set of all tracks over a finite Kripke structure $\mathcal{K}$. For any track $\rho = v_1 \cdots v_n \in \mathrm{Trk}_{\mathcal{K}}$, we define:

- $|\rho| = n$, $\mathrm{fst}(\rho) = v_1$, and $\mathrm{lst}(\rho) = v_n$;
- any index $i \in [1, |\rho|]$ is called a *ρ-position* and $\rho(i) = v_i$;
- $\mathrm{states}(\rho) = \{v_1, \cdots, v_n\} \subseteq W$;
- $\rho(i, j) = v_i \cdots v_j$, for $1 \leq i \leq j \leq |\rho|$, is the subtrack of ρ bounded by i, j;
- $\mathrm{Pref}(\rho) = \{\rho(1, i) \mid 1 \leq i \leq |\rho| - 1\}$ and $\mathrm{Suff}(\rho) = \{\rho(i, |\rho|) \mid 2 \leq i \leq |\rho|\}$ are the sets of all proper prefixes and suffixes of ρ, respectively.

Given $\rho, \rho' \in \mathrm{Trk}_{\mathcal{K}}$, we denote by $\rho \cdot \rho'$ the concatenation of the tracks ρ and ρ'. Moreover, if $\mathrm{lst}(\rho) = \mathrm{fst}(\rho')$, we denote by $\rho \star \rho'$ the track $\rho(1, |\rho| - 1) \cdot \rho'$. In particular, when $|\rho| = 1$, $\rho \star \rho' = \rho'$. In the following, when we write $\rho \star \rho'$, we implicitly assume that $\mathrm{lst}(\rho) = \mathrm{fst}(\rho')$. Finally, if $\mathrm{fst}(\rho) = w_0$ (the initial state of $\mathcal{K}$), ρ is called an *initial track*.

An abstract interval model (over $\mathrm{Trk}_{\mathcal{K}}$) can be naturally associated with a finite Kripke structure $\mathcal{K}$ by considering the set of intervals as the set of tracks of $\mathcal{K}$. Since $\mathcal{K}$ has loops (δ is left-total), the number of tracks in $\mathrm{Trk}_{\mathcal{K}}$, and thus the number of intervals, is infinite.

Definition 4. *The* abstract interval model induced by a finite Kripke structure $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$ *is* $\mathcal{A}_{\mathcal{K}} = (\mathcal{AP}, \mathbb{I}, A_{\mathbb{I}}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$, *where* $\mathbb{I} = \mathrm{Trk}_{\mathcal{K}}$, $A_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \mathrm{lst}(\rho) = \mathrm{fst}(\rho')\}$, $B_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \mathrm{Pref}(\rho)\}$, $E_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \mathrm{Suff}(\rho)\}$, *and* $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$ *is such that* $\sigma(\rho) = \bigcap_{w \in \mathrm{states}(\rho)} \mu(w)$, *for all* $\rho \in \mathbb{I}$.

Relations $A_{\mathbb{I}}, B_{\mathbb{I}}$, and $E_{\mathbb{I}}$ are interpreted as the Allen's relations A, B, and E, respectively. Moreover, according to the definition of σ, $p \in \mathcal{AP}$ holds over $\rho = v_1 \cdots v_n$ if and only if it holds over all the states $v_1, \cdots, v_n$ of ρ. This conforms to the *homogeneity principle* [19], according to which a proposition letter holds over an interval if and only if it holds over all its subintervals.

Definition 5. *Let $\mathcal{K}$ be a finite Kripke structure and ψ be an HS formula; we say that a track $\rho \in \mathrm{Trk}_{\mathcal{K}}$ satisfies ψ, denoted as $\mathcal{K}, \rho \models \psi$, iff it holds that $\mathcal{A}_{\mathcal{K}}, \rho \models \psi$. Moreover, we say that $\mathcal{K}$ models ψ, denoted as $\mathcal{K} \models \psi$, iff for all* initial *tracks $\rho' \in \mathrm{Trk}_{\mathcal{K}}$ it holds that $\mathcal{K}, \rho' \models \psi$. The* model checking problem *for HS over finite Kripke structures is the problem of deciding whether $\mathcal{K} \models \psi$.*

In Fig. 2, we provide an example of a finite Kripke structure $\mathcal{K}_{Sched}$ that models the behaviour of a scheduler serving three processes which are continuously requesting the use of a common resource (it is a simplified version of an example given in [13]). The initial state is v_0: no process is served in that state. In any other state v_i and $\overline{v}_i$, with $i \in \{1, 2, 3\}$, the i-th process is served (this is denoted by the fact that p_i holds in those states). For the sake of readability, edges are marked either by r_i, for *request*(i), or by u_i, for *unlock*(i). Edge labels do not have a semantic value, that is, they are neither part of the structure definition, nor proposition letters; they are simply used to ease reference to edges. Process i is served in state v_i, then, after "some time", a transition u_i from v_i to $\overline{v}_i$ is taken; subsequently, process i cannot be served again immediately, as v_i is not

directly reachable from $\overline{v}_i$ (the scheduler cannot serve the same process twice in two successive rounds). A transition r_j, with $j \neq i$, from $\overline{v}_i$ to v_j is then taken and process j is served.

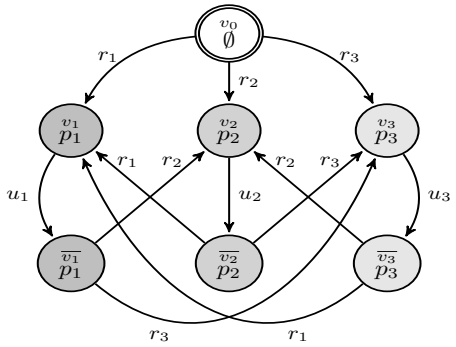


Fig. 2. The Kripke structure $\mathcal{K}_{Sched}$.

We now show how some meaningful properties to be checked against $\mathcal{K}_{Sched}$ can be expressed in HS, in particular, by formulas of $\mathsf{A\overline{A}E\overline{BE}}$. In all formulas, we force the validity of the considered property over all legal computation sub-intervals by using modality $[E]$ (all computation sub-intervals are suffixes of at least one initial track). The truth of the next statements can easily be checked ($\langle \mathrm{E} \rangle^k$ stands for k occurrences of modality $\langle \mathrm{E} \rangle$):

- $\mathcal{K}_{Sched} \models [E]\big(\langle \mathrm{E} \rangle^3 \top \rightarrow (\chi(p_1, p_2) \vee \chi(p_1, p_3) \vee \chi(p_2, p_3))\big)$, where $\chi(p, q) := \langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle p \wedge \langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle q$;
- $\mathcal{K}_{Sched} \not\models [E](\langle \mathrm{E} \rangle^{10} \top \rightarrow \langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle p_3)$;
- $\mathcal{K}_{Sched} \not\models [E](\langle \mathrm{E} \rangle^5 \rightarrow (\langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle p_1 \wedge \langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle p_2 \wedge \langle \mathrm{E} \rangle \langle \overline{\mathrm{A}} \rangle p_3))$.

The first formula states that in any suffix of length at least 4 of an initial track, at least 2 proposition letters are witnessed. $\mathcal{K}_{Sched}$ satisfies the formula since a process cannot be executed twice in a row. The second formula states that in any suffix of length at least 11 of an initial track, process 3 is executed at least once in some internal states (*non starvation*). $\mathcal{K}_{Sched}$ does not satisfy the formula since the scheduler can avoid executing a process ad libitum. The third formula states that in any suffix of length at least 6 of an initial track, p_1, p_2, p_3 are all witnessed. The only way to satisfy this property is to constrain the scheduler to execute the 3 processes in a strictly periodic manner, but this is not the case.

The general picture. Now we summarize the known complexity results about the MC problem for HS fragments (see Fig. 3 for a graphical account).

In [13], Molinari et al. show that, given a finite Kripke structure $\mathcal{K}$ and a bound k on the structural complexity of HS formulas (nesting depth of $\langle \mathrm{E} \rangle$ and $\langle \mathrm{B} \rangle$ modalities), it is possible to obtain a *finite* representation for $\mathcal{A}_{\mathcal{K}}$, which is equivalent to $\mathcal{A}_{\mathcal{K}}$ w. r. to satisfiability of HS formulas with structural complexity less than or equal to k. Then, by exploiting such a representation, they prove that the MC problem for (full) HS is decidable, providing an algorithm with non-elementary complexity. In [2], Bozzelli et al. show that the problem for the fragment BE, and thus for full HS, is **EXPSPACE**-hard. In [15], Molinari et al. study the fragments $\mathsf{A\overline{A}B\overline{BE}}$ and $\mathsf{A\overline{A}E\overline{BE}}$, devising for each of them an **EXPSPACE** MC algorithm which exploits the possibility of finding, for each track of a Kripke structure, a satisfiability-preserving track of bounded length (*track representative*). In this way, the algorithm needs to check only tracks having a bounded maximum length. In [14], they prove that the problem for

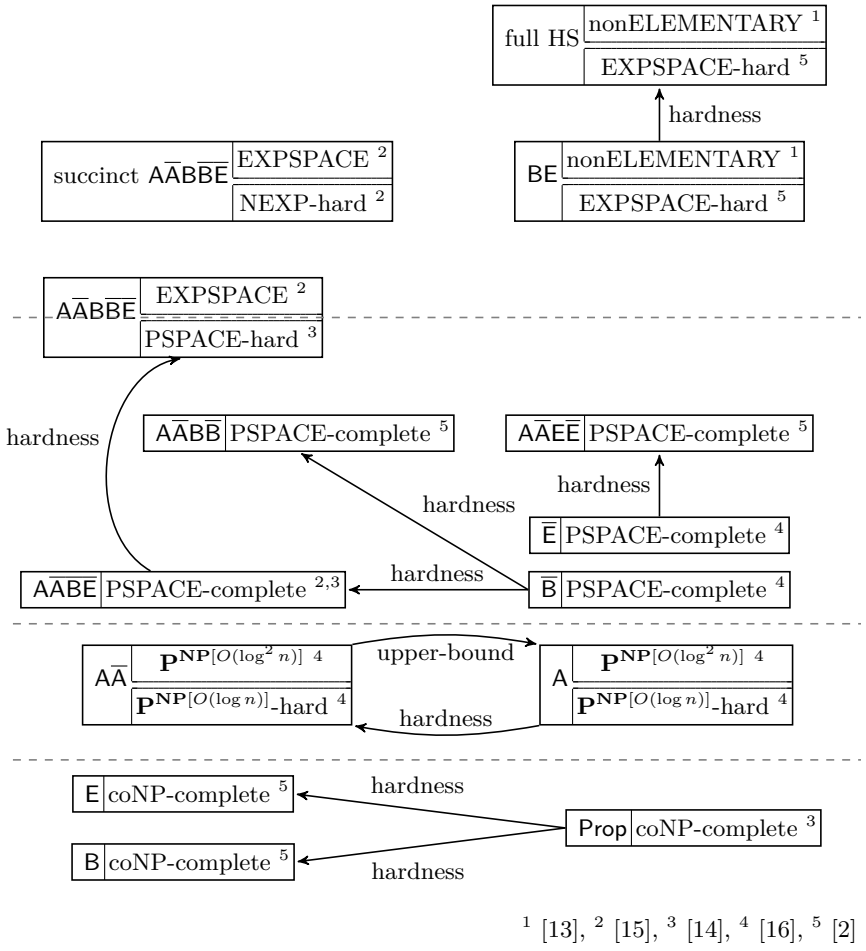


Fig. 3. Complexity of the MC problem for HS fragments

$\mathsf{A\overline{A}B\overline{B}\overline{E}}$ and $\mathsf{A\overline{A}E\overline{B}\overline{E}}$ is **PSPACE**-hard (with a succinct encoding of formulas the algorithm remains in **EXPSPACE**, but a **NEXPTIME** lower bound can be given [15]). The MC problem for other HS fragments has been studied in the following papers:

- $\mathsf{A\overline{A}B\overline{E}}$, $\mathsf{\overline{B}}$, $\mathsf{\overline{E}}$, $\mathsf{A\overline{A}B\overline{B}}$, and $\mathsf{A\overline{A}E\overline{E}}$ are **PSPACE**-complete [2, 14, 15, 16];
- $\mathsf{A\overline{A}}$, A, and $\mathsf{\overline{A}}$ are in between $\mathbf{P}^{\mathbf{NP}[O(\log n)]}$ and $\mathbf{P}^{\mathbf{NP}[O(\log^2 n)]}$ [16];
- B, E, Prop (the propositional fragment of HS) are **co-NP**-complete [2, 14].

In the next sections, we shall reconsider the MC problem for the fragment $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ (and the symmetric fragment $\mathsf{A\overline{A}E\overline{B}\overline{E}}$), proving in a much simpler way (compared to [15]) its membership to **EXPSPACE**. We shall show that, given a track ρ and $h \geq 0$, there is a track ρ', whose length is at most $(|W|+2)^{h+2}$, such that for every $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ formula ψ, with $\mathrm{d_B}(\psi) \leq h$, $\mathcal{K}, \rho \models \psi$ iff $\mathcal{K}, \rho' \models \psi$.

3 Track Bisimilarity

In this short section, we introduce the notions of *prefix-bisimilarity* and *suffix-bisimilarity* between a pair of tracks ρ and ρ' of a Kripke structure. As proved by Proposition 2 below, prefix-bisimilarity (resp., suffix-bisimilarity) is a sufficient condition for two tracks ρ and ρ' to be indistinguishable with respect to satisfiability of (some classes of) $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ (resp., $\mathsf{A\overline{A}E\overline{B}\overline{E}}$) formulas, respectively.

Definition 6 (Prefix-bisimilarity and Suffix-bisimilarity). *Let $h \geq 0$ and ρ and ρ' be two tracks of a Kripke structure $\mathcal{K}$. We say that ρ and ρ' are* h-prefix bisimilar *if the following conditions inductively hold:*

- *for $h = 0$:* $\mathrm{fst}(\rho) = \mathrm{fst}(\rho')$, $\mathrm{lst}(\rho) = \mathrm{lst}(\rho')$, *and* $\mathrm{states}(\rho) = \mathrm{states}(\rho')$.
- *for $h > 0$: ρ and ρ' are 0-prefix bisimilar and for each proper prefix ν of ρ (resp., proper prefix ν' of ρ'), there exists a proper prefix ν' of ρ' (resp., proper prefix ν of ρ) such that ν and ν' are $(h-1)$-prefix bisimilar.*

The notion of h-suffix bisimilarity *is defined in a symmetric way by considering suffixes of tracks instead of prefixes.*

Property 1. Given a Kripke structure $\mathcal{K}$, for all $h \geq 0$, h-prefix (resp., h-suffix) bisimilarity is an equivalence relation over $\mathrm{Trk}_{\mathcal{K}}$.

Moreover, h-suffix bisimilarity and h-prefix bisimilarity propagate downwards.

Property 2. Given a Kripke structure $\mathcal{K}$ and two tracks $\rho, \rho' \in \mathrm{Trk}_{\mathcal{K}}$, for all $h > 0$, if ρ and ρ' are h-prefix (resp., h-suffix) bisimilar, then they are also $(h-1)$-prefix (resp., $(h-1)$-suffix) bisimilar.

The following result can easily be proved by induction on $h \geq 0$.

Proposition 1. *Let $h \geq 0$, and ρ and ρ' be two h-prefix (resp., h-suffix) bisimilar tracks of a Kripke structure $\mathcal{K}$. Then, for each track ρ'' of $\mathcal{K}$,*

1. *$\rho'' \star \rho$ and $\rho'' \star \rho'$ are h-prefix (resp., h-suffix) bisimilar;*
2. *$\rho \star \rho''$ and $\rho' \star \rho''$ are h-prefix (resp., h-suffix) bisimilar.*

By Proposition 1 and a straightforward induction on the structural complexity of formulas, we obtain that h-prefix (resp., h-suffix) bisimilarity preserves the satisfiability of $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ (resp., $\mathsf{A\overline{A}E\overline{B}\overline{E}}$) formulas having nesting depth of modality B (resp., E) at most h.

Proposition 2. *Let $h \geq 0$, and ρ and ρ' be two h-prefix (resp., h-suffix) bisimilar tracks of a Kripke structure $\mathcal{K}$. For each $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ (resp., $\mathsf{A\overline{A}E\overline{B}\overline{E}}$) formula ψ with $\mathrm{d_B}(\psi) \leq h$ (resp., $\mathrm{d_E}(\psi) \leq h$), it holds that $\mathcal{K}, \rho \models \psi$ iff $\mathcal{K}, \rho' \models \psi$.*

4 The Fragments $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ and $\mathsf{A\overline{A}E\overline{B}\overline{E}}$: Exponential-Size Model-Track Property

In this section, we focus on the fragment $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ (the case of $\mathsf{A\overline{A}E\overline{B}\overline{E}}$ is completely symmetric). We shall show how to determine a subset of positions of a track ρ

(a *prefix sampling* of ρ), starting from which it is possible to build another track ρ', of bounded exponential size, which is indistinguishable from ρ with respect to the fulfilment of $\mathsf{A\overline{A}B\overline{B}E}$ formulas up to a given nesting depth of modality B (*exponential-size model-track property*). We start by introducing the notions of *induced track*, *prefix-skeleton sampling*, and *h-prefix sampling*, and prove some related properties.

Definition 7 (Induced track). *Let ρ be a track of length n of a Kripke structure $\mathcal{K}$. A* track induced by *ρ is a track π of $\mathcal{K}$ such that there exists an increasing sequence of ρ-positions $i_1 < \ldots < i_k$, with $i_1 = 1$, $i_k = n$, and $\pi = \rho(i_1) \cdots \rho(i_k)$.*

Note that if π is induced by ρ, then $\mathrm{fst}(\pi) = \mathrm{fst}(\rho)$, $\mathrm{lst}(\pi) = \mathrm{lst}(\rho)$, and $|\pi| \leq |\rho|$ (in particular, $|\pi| = |\rho|$ iff $\pi = \rho$). Intuitively, a track induced by ρ is obtained by contracting ρ, namely, by concatenating some subtracks of ρ, provided that the resulting sequence is a track of $\mathcal{K}$ as well.

In the following, given a set I of natural numbers, by "two consecutive elements of I" we refer to a pair of elements $i, j \in I$ s.t. $i < j$ and $I \cap [i, j] = \{i, j\}$.

Definition 8 (Prefix-skeleton sampling). *Let ρ be a track of a Kripke structure $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$. Given two ρ-positions i and j, with $i \leq j$, the* prefix-skeleton sampling *of $\rho(i, j)$ is the* minimal *set P of ρ-positions in the interval $[i, j]$ satisfying: (i) $i, j \in P$; (ii) for each state $w \in W$ occurring along $\rho(i + 1, j - 1)$, the minimal position $k \in [i + 1, j - 1]$ such that $\rho(k) = w$ is in P.*

From Definition 8, it immediately follows that the prefix-skeleton sampling P of (any) track $\rho(i, j)$ is such that $|P| \leq |W| + 2$ and $i + 1 \in P$ whenever $i < j$.

Definition 9 (h-prefix sampling). *Let ρ be a track of a Kripke structure $\mathcal{K}$. For each $h \geq 1$, the* h-prefix sampling of *ρ is the* minimal *set P_h of ρ-positions inductively satisfying the following conditions:*

- *Base case: $h = 1$. P_1 is the prefix-skeleton sampling of ρ;*
- *Inductive step: $h > 1$. (i) $P_h \supseteq P_{h-1}$ and (ii) for all pairs of consecutive positions i, j in P_{h-1}, the prefix-skeleton sampling of $\rho(i, j)$ is in P_h.*

The following upper bound to the cardinality of prefix samplings holds.

Property 3. Let $h \geq 1$ and ρ be a track of a Kripke structure $\mathcal{K}$. The h-prefix sampling P_h of ρ is such that $|P_h| \leq (|W| + 2)^h$.

We now prove a technical lemma that will be used in the proof of Lemma 2.

Lemma 1. *Let $h \geq 1$, ρ be a track of $\mathcal{K}$, and i, j be two consecutive ρ-positions in the h-prefix sampling of ρ. Then, for all ρ-positions $n, n' \in [i + 1, j]$ such that $\rho(n) = \rho(n')$, it holds that $\rho(1, n)$ and $\rho(1, n')$ are $(h - 1)$-prefix bisimilar.*

Proof. The proof is by induction on $h \geq 1$.

- Base case: $h = 1$. The 1-prefix sampling of ρ is the prefix-skeleton sampling of ρ. Hence, being i and j consecutive positions in this sampling, for each position $k \in [i, j - 1]$, there is $\ell \leq i$ such that $\rho(\ell) = \rho(k)$. Since $\rho(n) = \rho(n')$, $\mathrm{states}(\rho(1, n)) = \mathrm{states}(\rho(1, n'))$, so $\rho(1, n)$ and $\rho(1, n')$ are 0-prefix bisimilar.

– Inductive step: $h > 1$. By definition of h-prefix sampling, there are two consecutive positions i', j' in the $(h-1)$-prefix sampling of ρ such that i, j are consecutive positions of the prefix-skeleton sampling of $\rho(i', j')$.
If $i = i'$, then $j = i + 1$, hence, being $n, n' \in [i + 1, j]$, we get that $n = n'$, and the result trivially holds.
Now, assume that $i \neq i'$, thus $i > i'$. As in the base case, we easily deduce that $\rho(1, n)$ and $\rho(1, n')$ are 0-prefix bisimilar. It remains to show that for each proper prefix ν of $\rho(1, n)$ (resp., proper prefix ν' of $\rho(1, n')$), there is a proper prefix ν' of $\rho(1, n')$ (resp., proper prefix ν of $\rho(1, n)$) such that ν and ν' are $(h-2)$-prefix bisimilar. Let us consider a proper prefix ν of $\rho(1, n)$ (the proof for the other direction is symmetric). Hence, $\nu = \rho(1, m)$ for some $m < n$. We distinguish two cases:
  - $m \leq i$. Hence $\rho(1, m)$ is a proper prefix of $\rho(1, n')$ and the result follows.
  - $m > i$: since i and j are consecutive positions of the prefix-skeleton sampling of $\rho(i', j')$, $i > i'$, and $m \in [i+1, j-1]$ (hence $m < j'$), there exists $m' \in [i' + 1, i]$ such that $\rho(m') = \rho(m)$ and m' is in the prefix-skeleton sampling of $\rho(i', j')$. Let $\nu' = \rho(1, m')$. Evidently ν' is a proper prefix of $\rho(1, n')$ (as $n' \geq i + 1$). Moreover, since $m, m' \in [i' + 1, j']$ and i', j' are consecutive positions in the $(h-1)$-prefix sampling of ρ, by the inductive hypothesis $\nu = \rho(1, m)$ and $\nu' = \rho(1, m')$ are $(h-2)$-prefix bisimilar. □

The next lemma and the following theorem show how to derive, from any track ρ of a Kripke structure, another track ρ', induced by ρ and h-prefix bisimilar to ρ, such that $|\rho'| \leq (|W| + 2)^{h+2}$. By Proposition 2, ρ' is indistinguishable from ρ w.r.t. the fulfilment of any $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ formula ψ with $\mathrm{d_B}(\psi) \leq h$.

In order to build ρ', we first compute the $(h + 1)$-prefix sampling P_{h+1} of ρ. Next, for all the pairs of consecutive ρ-positions $i, j \in P_{h+1}$, we consider a track induced by $\rho(i, j)$, with no repeated occurrences of any state, except at most the first and last ones (hence, it is no longer than $(|W| + 2)$). The track ρ' is just the ordered concatenation (by means of the $\star$ operator) of all these tracks. The aforementioned bound on $|\rho'|$ holds as, by Property 3, $|P_{h+1}| \leq (|W|+2)^{h+1}$. The following preparatory lemma states that ρ and ρ' are indeed h-prefix bisimilar.

Lemma 2. *Let $h \geq 1$, ρ be a track of $\mathcal{K}$, and $\rho' = \rho(i_1)\rho(i_2)\cdots\rho(i_k)$ be a track induced by ρ, where $1 = i_1 < i_2 < \ldots < i_k = |\rho|$ and $P_{h+1} \subseteq \{i_1, \ldots, i_k\}$, with P_{h+1} the $(h + 1)$-prefix sampling of ρ. Then, for all $j \in [1, k]$, $\rho'(1, j)$ and $\rho(1, i_j)$ are h-prefix bisimilar.*

Notice that, in particular, ρ and ρ' are h-prefix bisimilar.

Proof. Let $Q = \{i_1, \ldots, i_k\}$ (hence $P_{h+1} \subseteq Q$) and let $j \in [1, k]$. We prove by induction on j that $\rho'(1, j)$ and $\rho(1, i_j)$ are h-prefix bisimilar. As for the base case ($j = 1$), the result holds, since $i_1 = 1$.

Now assume that $j > 1$. We first show that $\rho(1, i_j)$ and $\rho'(1, j)$ are 0-prefix bisimilar. Clearly, $\rho(1) = \rho(i_1) = \rho'(1)$, $\rho(i_j) = \rho'(j)$, and $\mathrm{states}(\rho'(1, j)) \subseteq \mathrm{states}(\rho(1, i_j))$. Now, if, by contradiction, there was a state w such that $w \in \mathrm{states}(\rho(1, i_j)) \setminus \mathrm{states}(\rho'(1, j))$, then for all $l \in Q$, with $l \leq i_j$, $\rho(l) \neq w$. However,

the prefix-skeleton sampling P_1 of ρ is contained in Q, and the minimal ρ-position l' such that $\rho(l') = w$ belongs to P_1. Since $w \in \text{states}(\rho(1, i_j))$, $l' \leq i_j$. Thus, we get a contradiction, implying that $\text{states}(\rho'(1, j)) = \text{states}(\rho(1, i_j))$.

It remains to prove that: (1) for each proper prefix ν' of $\rho'(1, j)$, there exists a proper prefix ν of $\rho(1, i_j)$ such that ν and ν' are $(h-1)$-prefix bisimilar, and (2) for each proper prefix ν of $\rho(1, i_j)$, there exists a proper prefix ν' of $\rho'(1, j)$ such that ν and ν' are $(h-1)$-prefix bisimilar.

As for (1), let ν' be a proper prefix of $\rho'(1, j)$. Hence, there exists $m \in [1, j-1]$ such that $\nu' = \rho'(1, m)$. By the inductive hypothesis, $\rho'(1, m)$ and $\rho(1, i_m)$ are h-prefix bisimilar, and thus $(h-1)$-prefix bisimilar as well (Property 2). Since $\rho(1, i_m)$ is a proper prefix of $\rho(1, i_j)$, by choosing $\nu' = \rho(1, i_m)$ (1) follows.

As for (2), assume that ν is a proper prefix of $\rho(1, i_j)$. Therefore, there exists $n \in [1, i_j - 1]$ such that $\nu = \rho(1, n)$. We distinguish two cases:

- $n \in P_{h+1}$. Since $n < i_j$, there exists $m \in [1, j-1]$ such that $n = i_m$. By the inductive hypothesis, $\rho(1, n)$ and $\rho'(1, m)$ are h-prefix bisimilar, and thus $(h-1)$-prefix bisimilar as well (Property 2). Since $\rho'(1, m)$ is a proper prefix of $\rho'(1, j)$, by choosing $\nu' = \rho'(1, m)$ (2) follows.
- $n \notin P_{h+1}$. It follows that there exist two consecutive positions i' and j' in P_{h+1}, with $i' < j'$, such that $n \in [i'+1, j'-1]$. By definition of $(h+1)$-prefix sampling, there exist two consecutive positions i'' and j'' in the h-prefix sampling of ρ, with $i'' < j''$, such that i' and j' are two consecutive positions in the prefix-skeleton sampling of $\rho(i'', j'')$.
  First, we observe that $i' \neq i''$ (otherwise, $j' = i' + 1$, which contradicts the fact that $[i'+1, j'-1] \neq \emptyset$, as $n \in [i'+1, j'-1]$). Thus, by definition of prefix-skeleton sampling applied to $\rho(i'', j'')$, and since $n \in [i'+1, j'-1]$, there must be $\ell \in [i''+1, i']$ such that $\rho(\ell) = \rho(n)$ and ℓ is in the prefix-skeleton sampling of $\rho(i'', j'')$. Hence $\ell \in P_{h+1}$ by definition of $(h+1)$-prefix sampling. As a consequence, since $\ell < n < i_j$, there exists $m \in [1, j-1]$ such that $\ell = i_m$. By applying Lemma 1, we deduce that $\rho(1, n)$ and $\rho(1, i_m)$ are $(h-1)$-prefix bisimilar. Moreover, by the inductive hypothesis, $\rho(1, i_m)$ and $\rho'(1, m)$ are $(h-1)$-prefix bisimilar. Thus, by choosing $\nu' = \rho'(1, m)$, ν' is a proper prefix of $\rho'(1, j)$ which is $(h-1)$-prefix bisimilar to $\nu = \rho(1, n)$. □

Theorem 1 (Exponential-size model-track property for $\mathsf{A\overline{A}B\overline{B}\overline{E}}$). *Let ρ be a track of a Kripke structure $\mathcal{K}$ and $h \geq 0$. Then, there exists a track ρ' induced by ρ, whose length is at most $(|W|+2)^{h+2}$, such that for every $\mathsf{A\overline{A}B\overline{B}\overline{E}}$ formula ψ with $\mathrm{d_B}(\psi) \leq h$, it holds that $\mathcal{K}, \rho \models \psi$ iff $\mathcal{K}, \rho' \models \psi$.*

Proof. Let P_{h+1} be the $(h+1)$-prefix sampling of ρ. For all pairs of consecutive ρ-positions i and j in P_{h+1}, there exists a track induced by $\rho(i, j)$ having length at most $|W|+2$, featuring no repeated occurrences of any internal state. We now define ρ' as the track of $\mathcal{K}$ obtained by concatenating in order all these induced tracks by means of the $\star$ operator. It is immediate to see that $\rho' = \rho(i_1)\rho(i_2)\cdots\rho(i_k)$, for some indexes $1 = i_1 < i_2 < \cdots < i_k = |\rho|$, where $\{i_1, \ldots, i_k\}$ contains the $(h+1)$-prefix sampling P_{h+1} of ρ. It holds that $|\rho'| \leq |P_{h+1}| \cdot (|W|+2)$ and since, by Property 3, $|P_{h+1}| \leq (|W|+2)^{h+1}$, we obtain that $|\rho'| \leq (|W|+2)^{h+2}$.

Moreover, by Lemma 2, ρ and ρ' are h-prefix bisimilar. By Proposition 2, the result follows. □

Theorem 1 allows us to easily devise an **EXPSPACE** MC algorithm for $\mathsf{A\overline{A}B\overline{B}E}$ formulas (and symmetrically for $\mathsf{A\overline{A}E\overline{B}E}$ formulas) which is basically the same as that presented in [15]. However, in that paper, the authors prove—in a much more involved way—the existence of a bound on the length of equivalent induced tracks which is greater than the present one, that is, $O(|W|^{2h+4})$.

5 Conclusions and Future Work

In this paper, we dealt with the problem of finding bounded representatives of tracks of a Kripke structure to solve the MC problem for the HS fragments $\mathsf{A\overline{A}B\overline{B}E}$ and $\mathsf{A\overline{A}E\overline{B}E}$. The proposed solution slightly reduces the bounds for track representatives given for the same problem in [15]; moreover, it substantially simplifies the constructions and the complexity of the proofs. As for future work, we would like to precisely characterize the complexity of MC for $\mathsf{A\overline{A}B\overline{B}E}$ and $\mathsf{A\overline{A}E\overline{B}E}$. At the moment, we only know that it belongs to **EXPSPACE** and it is **PSPACE**-hard [14]. More generally, we are looking for possible improvements to known complexity results for MC of (full) HS. We know that it is **EXPSPACE**-hard (we proved **EXPSPACE**-hardness of its fragment BE [2]), while the only available decision procedure is nonelementary [13].

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
2. Bozzelli, L., Molinari, A., Montanari, A., Peron, A., Sala, P.: Interval Temporal Logic Model Checking: the Border Between Good and Bad HS Fragments. In: IJCAR. pp. 389–405. LNAI 9706, Springer (2016)
3. Bresolin, D., Della Monica, D., Goranko, V., Montanari, A., Sciavicco, G.: The dark side of interval temporal logic: marking the undecidability border. Annals of Mathematics and Artificial Intelligence 71(1-3), 41–83 (2014)
4. Bresolin, D., Goranko, V., Montanari, A., Sala, P.: Tableau-based decision procedures for the logics of subinterval structures over dense orderings. Journal of Logic and Computation 20(1), 133–166 (2010)
5. Bresolin, D., Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. Annals of Pure and Applied Logic 161(3), 289–304 (2009)
6. Emerson, E.A., Halpern, J.Y.: "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. Journal of the ACM 33(1), 151–178 (1986)
7. Halpern, J.Y., Shoham, Y.: A propositional modal logic of time intervals. Journal of the ACM 38(4), 935–962 (1991)
8. Lodaya, K.: Sharpening the undecidability of interval temporal logic. In: ASIAN. pp. 290–298. LNCS 1961, Springer (2000)
9. Lomuscio, A., Michaliszyn, J.: An epistemic Halpern-Shoham logic. In: IJCAI. pp. 1010–1016 (2013)

10. Lomuscio, A., Michaliszyn, J.: Decidability of model checking multi-agent systems against a class of EHS specifications. In: ECAI. pp. 543–548 (2014)
11. Lomuscio, A., Michaliszyn, J.: Model checking multi-agent systems against epistemic HS specifications with regular expressions. In: KR. pp. 298–308 (2016)
12. Marcinkowski, J., Michaliszyn, J.: The undecidability of the logic of subintervals. Fundamenta Informaticae 131(2), 217–240 (2014)
13. Molinari, A., Montanari, A., Murano, A., Perelli, G., Peron, A.: Checking interval properties of computations. Acta Informatica (2016), accepted for publication.
14. Molinari, A., Montanari, A., Peron, A.: Complexity of ITL model checking: some well-behaved fragments of the interval logic HS. In: TIME. pp. 90–100 (2015)
15. Molinari, A., Montanari, A., Peron, A.: A model checking procedure for interval temporal logics based on track representatives. In: CSL. pp. 193–210 (2015)
16. Molinari, A., Montanari, A., Peron, A., Sala, P.: Model Checking Well-Behaved Fragments of HS: the (Almost) Final Picture. In: KR. pp. 473–483 (2016)
17. Moszkowski, B.: Reasoning About Digital Circuits. Ph.D. thesis, Dept. of Computer Science, Stanford University, Stanford, CA (1983)
18. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57 (1977)
19. Roeper, P.: Intervals and tenses. Journal of Philosophical Logic 9, 451–469 (1980)
20. Venema, Y.: Expressiveness and completeness of an interval tense logic. Notre Dame Journal of Formal Logic 31(4), 529–547 (1990)

Types for Immutability and Aliasing Control

Paola Giannini[1], Marco Servetto[2], and Elena Zucca[3]

[1] Università del Piemonte Orientale, Italy
giannini@di.unipmn.it
[2] Victoria University of Wellington, New Zealand
marco.servetto@ecs.vuw.ac.nz
[3] Università di Genova, Italy
elena.zucca@unige.it

1 Introduction

In mainstream languages with state and explicit mutations, unwanted aliasing relations are common bugs. This is exasperated by concurrency mechanisms, since unpredicted aliasing can induce unplanned/unsafe communication points between threads.

For these reasons, a massive amount of research, see, e.g., [16,9,11,6], has been devoted to make programming with side-effects easier to maintain and understand, notably using *type modifiers* to control state access. In this paper, we will use five type modifiers (`mut`, `imm`, `capsule`, `lent`, `read`) to obtain a fine-tuned control of immutability and aliasing properties.

Let us consider the store as a graph, where nodes contain records of fields, that may be references to other nodes. Each node determines a subgraph, that of all the nodes reachable from it. Let x be a reference to a node in the graph. Is x is *mutable*, then it can be freely used, and we cannot make any assumption on it. If x is *immutable*, then $x.f{=}e$ is not allowed, and we can assume that the subgraph reachable from x will not be modified through any other reference. An immutable reference can be safely shared also in a multithreaded environment. If x is a *capsule*, then we can assume that the subgraph reachable from x is an isolated portion of store, that is, all its (non immutable) nodes cannot be reached through other references. Capsule references can be used only once, as both mutable or immutable, e.g., to initialize a mutable/immutable reference. In other words, a capsule reference can be seen as a reference whose destiny has not been decided yet. Moreover, if a capsule reference x is assigned to a mutable one y, then (in that moment) no part of this subgraph can be updated through a reference different from y. This allows to identify mutable state that can be safely handled by a thread. If x is *lent*, then it can be used in a restricted way, so that

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 62–74 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

no aliasing can be introduced that would make possible to reach the subgraph of x. Finally, *readable* references can neither be modified nor aliased. These last two modifiers ensure intermediate properties used to derive the immutable and capsule properties.

Whereas (variants of) such modifiers have appeared in previous literature (see Sect.4 for a discussion on related work), there are two key novelties w.r.t. similar proposals. First, the expressivity of the type system is greatly enhanced. Indeed, modifiers restrict, as described above, the use of a reference in a context, but this restriction can be escaped by *promotion*, at the price of restricting the use of other references. Second, it is possible to express and check aliasing and immutability property directly on source terms, without introducing invariants on an auxiliary structure which mimics physical memory. Indeed, we adopt an innovative execution model [12,4,14] for imperative languages which, differently from traditional ones, is a *pure calculus*.

In this extended abstract, due to space constraints, we focus on the type system, and only illustrate the calculus by simple examples. We refer to the full paper [8] for reduction rules, more examples and proofs of results.

The rest of the paper is organized as follows: we provide syntax and an informal introduction in Sect.2, formalize the type system and state results in Sect.3, discuss related work, paper contribution and further work in Sect.4.

2 Informal introduction

Syntax and types are given in Fig.1. We assume sets of *variables* x, y, z, ..., *class names* C, *field names* f, and *method names* m. We adopt the convention that a metavariable which ends by s is implicitly defined as a (possibly empty) sequence, for example xs is defined by $xs ::= \epsilon \mid x\ xs$, where ϵ denotes the empty sequence.

cd	::= `class` C `{`$fds\ mds$`}`	class declaration
fd	::= `imm` $C\ f$ | `mut` $C\ f$ | `int`	field declaration
md	::= $T\ m\ \mu$ `(`$T_1\ x_1, \ldots, T_n\ x_n$`)` `{return` e`}`	method declaration
e	::= x | $e.f$ | $e.m$`(`es`)` | $e.f$`=`e | `new` C`(`es`)` | `{`$ds\ e$`}`	expression
d	::= $T\,x$`=`e;	variable declaration
T	::= $\mu\ C$ | `int`	type
μ	::= `imm` | `mut` | `capsule` | `lent` | `read`	type modifier
dv	::= $T\,x$`=`v;	evaluated declaration
u,v	::= x | `new` C`(`xs`)` | `{`$dvs\ x$`}` | `{`dvs `new` C`(`xs`)}`	value

Fig. 1: Syntax and types

The syntax mostly follows Java and Featherweight Java (FJ) [10]. A class declaration consists of a class name, a sequence of field declarations and a sequence

of method declarations. A field declaration consists of a field type and a field name. A method declaration consists, as in FJ, of a return type, a method name, a list of parameter names with their types, and a body which is an expression. However, there is an additional component: the type modifier for `this`, which is placed after the method name. As in FJ, we assume for each class a canonical constructor whose parameter list exactly corresponds to the class fields, and we assume no multiple declarations of classes in a class table, fields and methods in a class declaration.

For expressions, in addition to the standard constructs of imperative OO languages, we have *blocks*, which are sequences of variable declarations, followed by a *body* which is an expression. Variable declarations consist of a type, a variable and an initialization expression. Types are class names decorated by a *type modifier*. We also include `int` as an example of primitive type, but we do not formally model related operators used in the examples, such as integer constants and sum. We assume no multiple declarations for variables in a block.

Values are references x, *object states*, that is, constructor invocations where arguments are references, or blocks in which declarations are evaluated, that is, initialization expressions are values, and the body is a reference or an object state.

Blocks are a fundamental construct of our language, since sequences of local variable declarations, when evaluated, are used to directly represent store in the language itself. For instance[4], assuming that class `B` has a `mut` field of type `B`:

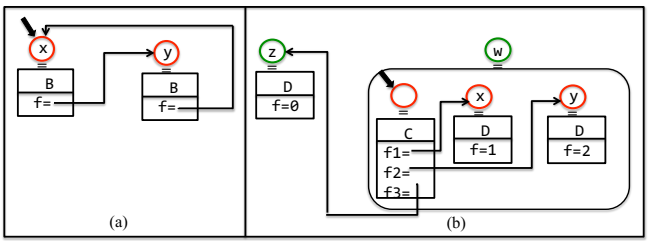


Fig. 2: Graphical representation of the store

```
mut B x= new B(y);    mut B y= new B(x);    x
```

the two declarations can be seen as a store where `x` denotes an object of class `B` whose field is `y`, and conversely, as shown in Fig.2(a). The whole block denotes a store with an entry point (graphically represented by a thick arrow), that is, an object.

Moreover, store is *hierarchical*, rather than flat as it usually happens in models of imperative languages. For instance, assuming that class `C` has two `mut` and one `imm D` fields, and class `D` has an integer field, the following is a store:

```
imm D z= new D(0);
imm C w= {mut D x= new D(1); mut D y= new D(2); new C(x,y,z)}
```

Here, the value associated to `w` is a block introducing local declarations, that is, in turn a store, as shown in Fig.2(b). The advantage of this hierarchical shape is that it models in a simple and natural way constraints about aliasing among objects, notably:

[4] In the examples, we omit for readability the brackets of the outermost block.

– the fact that an object is not referenced from outside some enclosing object is directly modeled by the block construct: for instance, the objects denoted by `x` and `y` can only be reached through `w`
– conversely, the fact that an object does not refer to the outside is modeled by the fact that the corresponding block is closed, that is, has no free variables[5]: for instance, the object denoted by `w` is not closed, since it refers to the external object `z`.

In the graphical representation, variables circled in red are mutable references, whereas the ones circled in green are immutable. Note that the reference corresponding to `new C(x,y,z)` is anonymous. Note also that, in this example, mutable variables in the local store of `w` are not visible from the outside. This models in a natural way the fact that the portion of store denoted by `w` is indeed immutable, as will be detailed in the sequel.

We illustrate now the meaning of the modifiers `mut`, `imm`, and `capsule`. A mutable variable refers to a portion of store that can be modified during execution. For instance, the block

```
mut B x= new B(y);   mut B y= new B(x);   x.f = x
```

reduces to

```
mut B x= new B(x);   mut B y= new B(x);   x
```

We give a graphical representation of this reduction in Fig.3. In the graphical representation, we highlight in grey expressions which are not values. So in (a) the body of the block is the expression `x.f=x`, whose evaluation modifies the field `f` of `x`, and returns `x`. The result of the reduction is shown in (b).

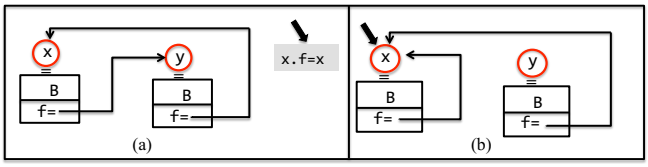


Fig. 3: Example of reduction (1)

Variables declared immutable, instead, once they have an associated value, cannot be modified. Immutability is *deep*, that is, all the nodes of the reachable object graph of an immutable reference are immutable themselves. Therefore, in the enclosing scope of the declaration

```
imm C w= {mut D x= new D(1); mut D y= new D(2); new C(x,y,z)}
```

the variable `z` must be declared `imm`, and we cannot have an assignment to a field of `w`.

A variable declared `capsule` refers to an *isolated* portion of store, where local objects can freely reference each other but for which the current variable is the only external reference. For instance:

```
capsule B z = { mut B x= new B(y);   mut B y= new B(x);   x }
```

[5] In other words, our calculus smoothly integrates memory representation with shadowing and α-conversion.

The internal objects denoted by `x` and `y` can be only be accessed through `z`. A capsule variable is a temporary reference, to be used once and for all to "move" an isolated portion of store to another node in the store. To get more flexibility, external immutable references are freely allowed. For instance, in the example above of the declaration of `w`, the inizialization expression has a `capsule` type. In our type system, capsule types are subtypes of both mutable and immutable types. Hence, capsule expressions can initialize both mutable and immutable references. However, to preserve the capsule property, we need a *linearity constraint*: in well-formed expressions capsule references can occur at most once in their scope.

Consider the term

```
mut D y=new D(0);
capsule C z={mut D x=new D(y.f=y.f+1); new C(x,x)}
```

In Fig.4(a) we have a graphical representation of this term, where the variable circled in blue is a capsule reference.

The evaluation of the expression on the right-hand side of `x` starts by evaluating `y.f+1`, which triggers the evaluation of `y.f`. The result is shown in (b), then the sum `0+1` is evaluated, returning `1`, as shown in (c). The evaluation of the field assignment `y.f=1`, updates the field `f` of `y` to `1`, and `1` is returned. Since `new D(1)` is a value, the whole term is fully evaluated, and it is shown in (d).

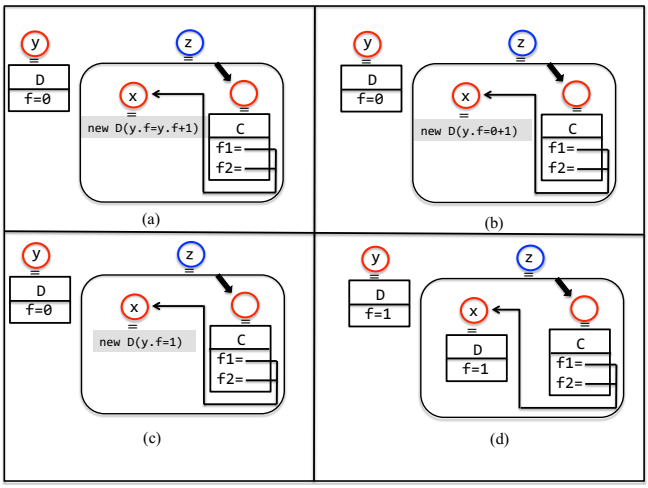


Fig. 4: Example of reduction (2)

To be able to typecheck more expressions as `capsule` or `imm`, we introduce the `lent` and `read` modifiers. References with such modifiers can be used in a restricted way. Notably, they can be used to access fields, but cannot be used either on the left or the right-hand side of an assignment or in an object creation. For `lent` references this restriction is not permanent, in the sense that it is possible to freely use a `lent` reference in a subexpression at the price of restricting other references, as will be detailed in the following section. Clearly, wherever a lent reference is required we could use a mutable one, so there is a subtyping relation between `lent` and `mut` modifiers. In some cases it is possible to move the type of an expression against the subtype hierarchy, that is, to *promote* an expression. Notably, a `mut` expression can be promoted to `capsule`, and a `read` expression can be promoted to `imm`, provided that some of the free variables in

the expression are used in a restricted way. The situation is graphically depicted in Fig.5. Promotions will be described in the next section.

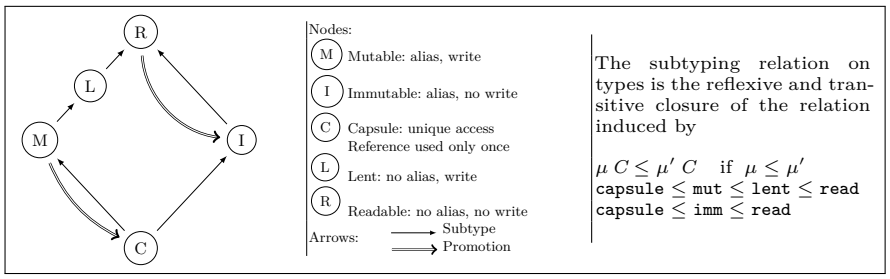


Fig. 5: Type modifiers and their relationships

3 Type system and results

A *type context* $\Delta = \Gamma; xss; ys$ consists of a usual *assignment of types* to variables $\Gamma = x_1{:}T_1, \ldots, x_n{:}T_n$, and two additional components xss and ys. Accordingly with our convention, xss is a sequence $xs_1 \ldots xs_n$ of sequences of variables, and ys is a sequence of variables. All such sequences are assumed to be sets (that is, order and repetitions are immaterial).

Sets $xs_1 \ldots xs_n$ are pairwise disjoint, and their elements are variables of `mut` type in Γ, hence they describe a partition of such variables in $n+1$ sets, called *groups*, the last being the set of those not belonging to any xs_i, called the *current group*. Elements of ys are variables of `mut`, `lent` or `read` type in Γ. The sets xss and ys model restrictions on the variables in Γ, as motivated below.

The type system must assure that, if for an expression e we derive type T, then the evaluation of the expression produces a value with the property expressed by T. For instance, if the following two variable declarations are well-typed `capsule B z1 = e1; imm B z2 = e2;` then the evaluation of the expressions `e1` and `e2` in the context of a well-typed program should produce values which are capsule and immutable, respectively, in the sense introduced in the previous section.

Let us discuss, for instance, when an expression could be safely typed `capsule`. Obviously, this is safe for an expression with no free variables, or where all the free variables are immutable. However, this requirement is too strong. For instance, consider the following sequence of declarations:

```
mut D y=new D(0);capsule C z={mut D x=new D(y.f); new C(x,x)};
```

The inner block (right-hand side of the declaration of `z`) can be typed `capsule`, even though there is a mutable free variable `y`, since this variable is only used in a field access. However, if we had `y` instead of `new D(y.f)`, the declaration would not be well-typed, since through the variable `y` we would refer to the reachable graph of `z`.

Formally, the inner block can be typechecked in a type context where *xss* consists of only one singleton group containing variable `y`. This means that this variable, which was originally `mut`, is restricted to `lent` when typechecking the inner block, hence no aliasing can be introduced between `y` and the final result of the block.

In general, in the *typing judgment* $\Gamma; xss; ys \vdash e : T$, the sets *xss* and *ys* model restrictions on aliasing which could be possibly introduced by the evaluation of the expression. That is, if *xs* and *xs′* are two different groups in *xss*, and if their reachable graphs were disjoint before the evaluation of the expression, then after the evaluation they would be still disjoint. Analogously, no aliasing can be introduced between the reachable graph of any group *xs* in *xss* and that of the result. The same constraint holds for the portion of store reachable from the variables in *ys*. These restrictions are achieved as follows:

- a `mut` variable in Γ which belongs to a group in *xss* is *lent-restricted*, that is, can only be used as `lent`
- a `read`, `lent` or `mut` variable in Γ which belongs to *ys* is *strongly-restricted*, that is, cannot be used at all.

Variables become restricted as an effect of applying *promotion* rules (T-CAPSULE) and (T-IMM), and restricted variables can be temporarily unrestricted by applying (T-SWAP) and (T-UNRST) rules, as will be explained in detail later.

Typing rules are given in Fig.6. In the rules we use information extracted from the class table, which is modelled, as usual, by the following functions:

- $\mathsf{fields}(C)$ gives, for each declared class C, the sequence of its fields declarations
- $\mathsf{method}(C, m)$ gives, for each method m declared in class C, the tuple $\langle T, \mu, T_1\ x_1 \ldots T_n\ x_n, e\rangle$ consisting of its return type, type modifier for `this`, parameters, and body.

We assume method bodies to be well-typed w.r.t. the type annotations in the method declaration. Formally, if
$\mathsf{method}(C, m) = \langle T, \mu, T_1\ x_1 \ldots T_n\ x_n, e\rangle$, then it should be $\Gamma; \emptyset; \emptyset \vdash e : T$, with $\Gamma = \texttt{this}{:}\mu\ C, x_1{:}T_1, \ldots, x_n{:}T_n$.

Rules (T-CAPSULE) and (T-IMM) model *promotions*, that is, can be used to promote an expression from a more general to a more specific type, under the conditions that some free variables in the expression are restricted. There are two kinds of promotion:

- `mut` ⇒ `capsule`
  As shown in rule (T-CAPSULE), an expression can be typed `capsule` in $\Gamma; xss; ys$ if it can be typed `mut` by lent-restricting the current group of mutable variables (*xs*). Formally, this group is added to *xss*.
- `read` ⇒ `imm`
  As shown in rule (T-IMM), an expression can be typed `imm` in $\Gamma; xss; ys$ if it can be typed `read` by strongly-restricting, rather than *ys* only, *all* currently available mutable, lent, and readable variables ($\mathsf{dom}^{\geq \texttt{mut}}(\Gamma)$). The current group of mutable variables (*xs*) is also added as a new group to *xss*.

Note that set difference in the side conditions makes sense since xss is assumed to be a set.

$$\text{(T-CAPSULE)}\ \frac{\Gamma; xss\ xs; ys \vdash e : \texttt{mut}\ C}{\Gamma; xss; ys \vdash e : \texttt{capsule}\ C}\ xs = \mathsf{dom}^{\texttt{mut}}(\Gamma) \backslash xss$$

$$\text{(T-IMM)}\ \frac{\Gamma; xss\ xs; \mathsf{dom}^{\geq \texttt{mut}}(\Gamma) \vdash e : \texttt{read}\ C}{\Gamma; xss; ys \vdash e : \texttt{imm}\ C}\ xs = \mathsf{dom}^{\texttt{mut}}(\Gamma) \backslash xss$$

$$\text{(T-SWAP)}\ \frac{\Gamma; xss\ xs'; ys \vdash e : \mu\ C}{\Gamma; xss\ xs; ys \vdash e : \mu'\ C}\ xs' = \mathsf{dom}^{\texttt{mut}}(\Gamma) \backslash (xss\ xs)\ \ \mu' = \begin{cases} \texttt{lent} & \text{if } \mu = \texttt{mut} \\ \mu & \text{otherwise} \end{cases}$$

$$\text{(T-UNRST)}\ \frac{\Gamma; xss; \emptyset \vdash e : \mu\ C}{\Gamma; xss; ys \vdash e : \mu\ C}\ \mu \leq \texttt{imm} \qquad \text{(T-SUB)}\ \frac{\Delta \vdash e : T}{\Delta \vdash e : T'}\ T \leq T'$$

$$\text{(T-VAR)}\ \frac{}{\Gamma; xss; ys \vdash x : T'}\ \Gamma(x) = T \ \wedge\ x \notin ys\ \ T' = \begin{cases} \texttt{lent}\ C & \text{if } T = \texttt{mut}\ C \wedge x \in xss \\ T & \text{otherwise} \end{cases}$$

$$\text{(T-FIELD-ACCESS)}\ \frac{\Delta \vdash e : \mu\ C}{\Delta \vdash e.f_i : T'_i}\ \mathsf{fields}(C) = T_1\ f_1 \ldots T_n\ f_n\ \ T'_i = \begin{cases} \mu\ C_i & \text{if } T_i = \texttt{mut}\ C_i \\ T_i & \text{otherwise} \end{cases}$$

$$\text{(T-METH-CALL)}\ \frac{\Delta \vdash e_i : T_i \quad \forall i \in 0..n}{\Delta \vdash e_0.m(e_1, \ldots, e_n) : T}\ T_0 = \mu\ C\ \ \mathsf{method}(C, m) = \langle T, \mu, T_1\ x_1 \ldots T_n\ x_n, e \rangle$$

$$\text{(T-FIELD-ASSIGN)}\ \frac{\Delta \vdash e : \texttt{mut}\ C \quad \Delta \vdash e' : T_i}{\Delta \vdash e.f_i = e' : T_i}\ \mathsf{fields}(C) = T_1\ f_1 \ldots T_n\ f_n$$

$$\text{(T-NEW)}\ \frac{\Delta \vdash e_i : T_i \quad \forall i \in 1..n}{\Delta \vdash \texttt{new}\ C(e_1, \ldots, e_n) : \texttt{mut}\ C}\ \mathsf{fields}(C) = T_1\ f_1 \ldots T_n\ f_n$$

$$\text{(T-BLOCK)}\ \frac{\Gamma[\Gamma']; xss'; ys \vdash e_i : T_i \quad \forall i \in 1..n \qquad \Gamma[\Gamma']; xss'; ys \vdash e : T}{\Gamma; xss; ys \vdash \{\, T_1\ x_1 = e_1; \ldots T_n\ x_n = e_n;\ e \,\} : T}\ \begin{array}{l} \Gamma' = x_1 : T_1, \ldots, x_n : T_n \\ xss = xss'_{|\mathsf{dom}^{\texttt{mut}}(\Gamma)} \end{array}$$

Fig. 6: Typing rules

Along with promotion rules, we have two rules which make it possible to temporarily unrestrict some of the restricted variables. In the detail:

– (T-SWAP) an expression can be typed in $\Gamma; xss\ xs; ys$ if it can be typed by unrestricting some group (xs) of lent-restricted variables, by swapping this group with the current group of mutable variables (xs'). The type obtained in this way is weakened to `lent`, if it was `mut`.

– (T-UNRST) an expression can be typed in $\Gamma; xss; ys$ if it can be typed by unrestricting all strongly-restricted variables ys, provided that the type obtained in this way is `capsule` or `imm`.

We illustrate in more detail typing rules (T-CAPSULE) and (T-SWAP) for `capsule` promotion. Since in the premises of rule (T-CAPSULE) all the mutable variables in Γ are lent-restricted, the reachable graph obtained by evaluating e will contain mutable references only to local variables, which cannot be accessed from outside their scope. So the modifier of the expression can be promoted to `capsule`. Note that, if, for the expression e, we derive a type with a `lent` modifier, then the result of the evaluation of e could be an external mutable reference, therefore the value would not be a capsule.

Consider now the case in which, in the evaluation of a capsule expression, we need to perform some field assignment, as in the example of Fig.4. Typing rule (T-CAPSULE) can be applied if the initialization expression of `z` can get type `mut C` in a context with type assignment `y:mut D,z: capsule C` and the group `y` of lent-restricted variables. However, the assignment `y.f=y.f+1` is not well-typed *in this type context*, since the variable `y` has type `lent D`. However, intuitively, we can see that the assignment does not introduce any alias between `y` and the final result, since it involves only variables which are in the same group (the singleton `y`), and produces an immutable result (of type `int`). So, it should be possible to promote the expression to a capsule.

To allow such typing, we introduce rule (T-SWAP), that removes the (lent) restriction on the variables of one of the groups, say xs, so that the variables in xs can be used as mutable, adding to xss a group xs', containing all the mutable variables in Γ which are not lent-restricted yet. In this way, we know that the evaluation of the expression will not introduce any alias between the variables in the swapped group and the current group of mutable variables. Moreover, if we derive, in this new context, an immutable or capsule type, we know that the result of the expression will be a value that can be freely used. In our example, we can apply rule (T-SWAP) when deriving the type for `y.f=y.f+1`, swapping the group `y` with `x`. Instead, if we derive a mutable type, then this type is weakened to `lent`, since the result could contain references to the variables in group xs', which were lent-restricted in the original context. This is shown by the following example:

```
mut D y=new D(x1,x2);  mut x1=new A(0); mut x2 = new A(1);
capsule C z={mut A x=(y.f1=y.f2); new C(x,x)};
```

If we apply rule (T-SWAP) when deriving the type for `y.f1=y.f2`, therefore swapping the group `y` with `x`, then we derive type `mut A`, and rule (T-SWAP) would assign type `lent A` to the expression. Therefore, the declaration `mut A x=(y.f1=y.f2)` and the whole expression would be ill-typed. Indeed, the expression reduces to

```
mut D y=new D(x2,x2);  mut x1=new A(0); mut x2 = new A(1);
capsule C z={mut A x=x2; new C(x,x)};
```

in which the value of `z` is not a capsule.

In rule (T-BLOCK), we write $\Gamma[\Gamma']$ for the concatenation of Γ and Γ' where, for the variables occurring in both domains, Γ' takes precedence. Moreover,

$xss_{|xs}$ denotes the sequence obtained from xss by keeping (in each element of the sequence) only the variables in xs. A block is well-typed if the right-hand sides of declarations and the body are well-typed w.r.t. a type context consisting of:

- the type assignment of the enclosing scope Γ, updated by that of the locally declared variables Γ'
- groups of lent-restricted variables xss, where a locally declared (mutable) variable can belong to any group, however preserving the partition in groups of the enclosing scope, as imposed by the second side condition
- the strongly-restricted variables ys of the enclosing scope.

Typechecking a block with some local variables lent-restricted in the same group of some variables of the enclosing scope can be necessary. Consider the following example

```
mut D z=new D(0);  mut C x=new C(z,z);
capsule D y={mut D z1=new D(1);lent D z2=(x.f1=z1); new D(1)};
```

Since we need to apply the promotion to capsule to the block on right-hand side of the declarations of `y`, the context of the typing of the block must be `z : mut D, x : mut C, y : capsule D; z x;` $\emptyset$, that is, `z` and `x` are in the same group of lent-restricted variables. However, to apply rule (T-FIELD-ASSIGN) to the expression `x.f1=z1`, both `x` and `z1` have to be mutable[6]. Therefore, we have to apply rule (T-SWAP), and it must be the case that both `x` and `z1` are in the same group of lent-restricted variables. This is possible, with rule (T-BLOCK), by adding `z1` to the group `x z`, in typing the right-hand sides of the declarations.

Other rules are mostly standard, except that they model the expected behaviour of type modifiers.

In rule (T-VAR), a variable is weakened to `lent` if it belongs to some group in xss, and cannot be used at all if it belongs to ys.

In rule (T-FIELD-ACCESS), in case the field is `mut`, the type modifier of the receiver is propagated to the field. For instance, mutable fields referred through a `lent` reference are `lent` as well. If the field is immutable, instead, then the expression has type `imm`, regardless of the receiver type.

In rule (T-FIELD-ASSIGN), the receiver should be mutable, and the right-hand side must have the field type. Note that this implies the right-hand side to be either `mut` or `imm` (or of a primitive type). Hence, neither the left-hand nor the right-hand sides can be `lent` or `read`, thus preventing the introduction of aliases.

In rule (T-NEW), analogously, expressions assigned to fields cannot be `lent`. Note that an object is created with no restrictions, that is, as `mut`.

Finally, note that primitive types are used in the standard way. For instance, in the premise of rule (T-NEW) the types of constructor arguments could be primitive types as well, whereas in rule (T-METH-CALL) the type of receiver could not.

Results The type system is sound for the operational semantics, that is, subject reduction and progress hold. Note that, since our operational model is a pure calculus, in the statements and proofs we do not need invariants on auxiliary structures such as memory.

[6] Assuming that field `f1` is mutable.

The reduction relation $e \longrightarrow e'$ is defined in the full paper [8]. We write $\vdash e : T$ for $\emptyset;\emptyset;\emptyset \vdash e : T$.

Theorem 1. *Let $\vdash e : T$. Then, either e is a value or $e \longrightarrow e'$ for some e' such that $\vdash e' : T$ is derivable.*

In addition to the standard soundness property, the `capsule` and `imm` modifiers have the expected behaviour, that is, in the evaluation of a well-typed expression,

- initialization expressions of `capsule` variables reduce to values whose *free variables are immutable*, and
- values associated to immutable variables *are not modified by the evaluation.*

The formalization and proof of these properties can be found in the full paper [8].

4 Related work and conclusion

A first, informal, version of our type modifiers has been presented in [13]. In [14] were introduced the `capsule` and `lent` modifiers, and a preliminary version of the promotion from mutable to capsule. The main novelties w.r.t. [14] are the introduction of the immutable and readable modifiers, the readable to immutable promotion, the formalization of the type system, and the proof of its properties.

Our type system combines in a novel and powerful way different features existing in previous work. Notably, the `capsule` notion has many variants in the literature, such as *external uniqueness* [5], *balloon* [1,13], *island* [7], *recovery* [9], and the fact that aliasing can be controlled by using *lent* (*borrowed*) references is well-known [11].

However, in our type system the `lent` notion is used in a novel way to achieve capsule promotion, since external mutable references are not forbidden once and for all as in recovery [9] but only lent-restricted, as illustrated by the last example in Sect.2.

Moreover, uniqueness is guaranteed by linearity, that is, by allowing at most one use of a `capsule` reference, rather than by destructive reads as in [9,3]. Destructive reads allows *isolated* fields, but has a serious drawback: an *isolated* field can become unexpectedly not available, hence any object contract involving such field can be broken.

Our `read` modifier is different from *readonly* as used, e.g., in [2]. An object cannot be modified through a readable/readonly reference. However, `read` also prevents static aliasing.

Javari [15] is a working backward-compatible extension of Java which also supports the *readonly* type modifier, and makes a huge design effort to support *assignable* and *mutable* fields, to have fine-grained readonly constraints. The need of such flexibility is motivated by performance reasons. In our design philosophy, we do not offer any way to a programmer of breaking the invariants enforced by the type system. Since our invariants are very strong, we expect compilers to

be able to perform optimization, thus recovering most of the efficiency lost to properly use immutable and readable objects.

Finally, the Pony language [6], providing an implementation as well, builds on the capabilities/recovery mechanisms of [9] as we do, but goes in a different direction, by distinguishing six different reference capabilities: *isolated* (similar to our capsule, but allowed in fields with destructive reads); *value* (similar to our immutable); *reference* (similar to our mutable); *box* (similar to readonly); *tag* (only allows object identity checks) and finally *transition* (a subtype of *box* that can be converted in *value*: a way to create values without using isolated references).

The key contributions of the paper are:

- A powerful type system for tracing mutation and aliasing in class-based imperative languages, providing: type modifiers for restricting the use of references; rules for promoting references to a less restrictive type at the price of restricting other references; rules for temporarily unrestricting references for typing subexpressions.
- A non standard operational model of the language as a pure calculus, relying on the language ability to represent cyclic object graphs.

This work is part of the development of L42, a novel programming language designed to support massive use of libraries, see the web site `L42.is`. The current L42 prototype is important as proof-of-evidence that the type system presented in this paper can be smoothly integrated with features of a realistic language. The prototype implements an algorithmic version of the type system which, roughly, attempts at applying promotions in all the possible ways, which are finitely many and, in practice, very few. A theoretical counterpart of such algorithmic version will be subject of further work. As a long term goal, we also plan to investigate (a form of) Hoare logic on top of our model. Finally, it should be possible to use our approach to enforce safe parallelism, on the lines of [9,13].

Acknowledgments. We are grateful to the anonymous reviewers for their useful suggestions, which led to substantial improvements.

References

1. Paulo Sérgio Almeida. Balloon types: Controlling sharing of state in data types. In *ECOOP'97* , volume 1241 of *LNCS*, pages 32–59. Springer.
2. Adrian Birka and Michael D. Ernst. A practical type system and language for reference immutability. In *OOPSLA 2004*, pages 35–49. ACM Press.
3. John Boyland. Semantics of fractional permissions with nesting. *ACM TOPLAS*, 32(6), 2010.
4. Andrea Capriccioli, Marco Servetto, and Elena Zucca. An imperative pure calculus. In *ICTCS'15*, ENTCS 322, 87–102, 2015.
5. David Clarke and Tobias Wrigstad. External uniqueness is unique enough. In *ECOOP'03*, volume 2473 of *LNCS*, pages 176–200. Springer.
6. Sylvan Clebsch, Sophia Drossopoulou, Sebastian Blessing, and Andy McNeil. Deny capabilities for safe, fast actors. In *AGERE! 2015*, pages 1–12. ACM Press.

7. Werner Dietl, Sophia Drossopoulou, and Peter Müller. Generic universe types. In *ECOOP'07*, volume 4609 of *LNCS*, pages 28–53. Springer.
8. Paola Giannini, Marco Servetto, and Elena Zucca. Transparent aliasing and mutation control. `http://people.unipmn.it/giannini/papers/ICTCS16-complete.pdf`.
9. Colin S. Gordon, Matthew J. Parkinson, Jared Parsons, Aleks Bromfield, and Joe Duffy. Uniqueness and reference immutability for safe parallelism. In *OOPSLA 2012*, pages 21–40. ACM Press.
10. Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *ACM TOPLAS*, 23(3):396–450, 2001.
11. Karl Naden, Robert Bocchino, Jonathan Aldrich, and Kevin Bierhoff. A type system for borrowing permissions. In *POPL 2012*, pages 557–570. ACM Press.
12. Marco Servetto and Lindsay Groves. True small-step reduction for imperative object-oriented languages. In *FTfJP'13*, pages 1–7. ACM Press.
13. Marco Servetto, David J. Pearce, Lindsay Groves, and Alex Potanin. Balloon types for safe parallelisation over arbitrary object graphs. In *WoDet 2013*.
14. Marco Servetto and Elena Zucca. Aliasing control in an imperative pure calculus. In *APLAS 15*, volume 9458 of *LNCS*, pages 208–228. Springer.
15. Matthew S. Tschantz and Michael D. Ernst. Javari: Adding reference immutability to Java. In *OOPSLA 2005*, pages 211–230. ACM Press.
16. Yoav Zibin, Alex Potanin, Paley Li, Mahmood Ali, and Michael D. Ernst. Ownership and immutability in generic Java. In *OOPSLA 2010*, pages 598–617. ACM Press.

Runtime checks as nominal types

Paola Giannini[1], Marco Servetto[2], and Elena Zucca[3]

[1] Università del Piemonte Orientale, Italy
giannini@di.unipmn.it
[2] Victoria University of Wellington, New Zealand
marco.servetto@ecs.vuw.ac.nz
[3] Università di Genova, Italy
elena.zucca@unige.it

Abstract. We propose a language design where types can be enriched by *tags* corresponding to predicates written by the programmer. For instance, `int&positive` is a type, where `positive` is a user-defined boolean function on integers. Expressions of type `int&positive` are obtained by an explicit *check* construct, analogous to cast, e.g., `(positive) 2`. In this way, the fact that the value of an expression is guaranteed to succeed a runtime check is a *static* property which can be controlled by the type system. We formalize our proposal as an extension of the simply-typed lambda calculus, and prove, besides soundness, the fact that expressions of tagged types reduce to values which satisfy the corresponding predicates.

1 Introduction

It is very common in programming that an application needs to handle only values which satisfy properties which can be checked through user-defined code. As a very simple example, integer numbers which are odd, or positive, or prime. In nominally typed languages, it is possible to declare specialized types which enforce/encode such invariants. This is obtained by performing runtime checks on the arguments of the constructor of the new type. There are two variations of this technique, the first is *wrapping* the original type, and the second is *extending* it. We describe them, by examples, using Java syntax.

The first solution is illustrated by the example of odd numbers:

```
class Odd{
  public final int inner;
  public Odd(int inner){
    if (inner%2==0){//not odd
      throw new Error(...); }
```


V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 75–87 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

```
    this.inner=inner;
  }
}
```

Here we encode by wrapping, in such a way that all instances of `Odd` will always contain an odd number. In particular, every method taking as input an `Odd` argument `myOdd` can rely on the (static) guarantee that `myOdd.inner` is an odd number.

The second solution is illustrated by an example of points with positive coordinates:

```
class Point{int x; int y;...}
class PositivePoint extends Point{
  public PositivePoint(int x, int y){
    super(x,y);//checks parent invariant
    if (x<0 || y<0){//not positive
      throw new Error(...); }
  }
}
```

Here we encode by extending the original type (class), calling the super constructor and then checking for more properties. Again, every method taking in input a `PositivePoint` may assume that the coordinates are positive numbers.

Both patterns could be verified by tools like Spec# [3] or Viper [9].[4]

Comparing the two solutions, we can say that:

- Wrapping can be always applied, while extending cannot be applied on primitive types and final classes.
- While extending implies subtyping, a wrapped value has not a subtype of the original type, thus forcing extra boilerplate code in the points of usage to access the inner value.
- On the other hand, wrapping can be applied to existing values, while extending requires the creation of a new value.
- Anyway, both techniques require some amount of boilerplate code.

We propose an alternative approach, whose goal is to synthesize the core concept behind these techniques: a user-defined predicate is checked on a value, and, in case of success, the value gets a more specific type, that keeps track of the success of the runtime test. For instance, we can declare two functions checking whether an integer is odd or positive, respectively:

```
bool odd(int i){return i%2!=0;}
bool positive(int i){return i>=0;}
```

[4] Viper requires to encode the invariant as constructor postcondition, and to repeat it in the method precondition, while Spec# needs the assertion to be explicitly stated in the class contract, and allows strengthening of superclass invariants (as needed in `PositivePoint`).

Then, we can convert an integer to an odd, or to a positive, by a *check* construct, which has a cast-inspired syntax, and indeed can be seen as a generalization of cast:[5]

```
(odd)35//ok
(positive)-23 //runtime error
(positive)(odd)79 //checks chain
```

The expressions above have, respectively, *tagged* types: `int&odd`, `int&positive`, and `int&odd&positive`. Tagged types are *nominal*, since tags are predicate names, and behave as intuitively expected: tag sequences can be seen as sets (order and repetition are immaterial), and subtyping corresponds to set inclusion. They can be used, e.g., as types of local variables:

```
int&positive&odd myInt=(positive)(odd)79
int&positive anotherInt=myInt //safe, thanks to subtyping
```

or as types of function parameters/result:

```
int&positive factorial(int&positive n){
  if (n==0){return 1;}
  return n * factorial((positive) n-1); }
```

Note that the function does not need to check whether the argument is positive (to avoid non termination), since this is ensured by the parameter type. However, we need the check to perform the recursive call.

The paper is organized as follows. We formalize our proposal and prove its expected properties in Sect.2. In Sect.3 we discuss related work, and present our conclusions and further work in Sect.4.

2 The calculus

In this section we introduce syntax, operational semantics, and type system of our calculus. Moreover, we prove expected properties.

Syntax We formalize our proposal as an extension of the simply-typed (call-by-value) lambda calculus. Syntax is given in Fig.1. We assume an infinite set of *predicate names* P.

Expressions include expressions of the lambda calculus (variable, abstraction and application) and those defined with operators of primitive types, e.g., the two boolean constants, and the infinite set of integer constants, ranged over by n. For simplicity we omit other standard additional constructs, such as conditional, let-in and recursion. In addition, there are two novel expressions:

- `(`P`)`e is a *check expression*, corresponding to a runtime check that (the value of) expression e satisfies predicate P. If the runtime check succeeds, then the result of the check expression will be the value of e tagged by P. In case of failure, the result will be an error tagged by P.

[5] That is, a standard cast expression `(C)e` can be seen as a particular case of check expression where the predicate is `instanceof C`.

p	$::= P\texttt{=}\lambda x : T.e$	predicate definition
e	$::=$	expression
	$x \mid \lambda x : T.e \mid e_1\ e_2$	
	$\mid \texttt{true} \mid \texttt{false} \mid n \mid \ldots$	
	$\mid \texttt{(}P\texttt{)}\, e$	check expression
	$\mid v\&P[e]$	checking expression
v	$::=$	base value
	$\texttt{true} \mid \texttt{false} \mid n \mid \ldots$	primitive value
	$\mid v\&P$	tagged value
V	$::= \lambda x : T.e \mid v$	value
$\mathcal{E}$	$::= [\,] \mid \mathcal{E}\ e \mid V\,\mathcal{E}$	evaluation context
	$\texttt{(}P\texttt{)}\mathcal{E} \mid v\&P[\mathcal{E}]$	
t	$::=$	base type
	$\texttt{bool} \mid \texttt{int} \mid \ldots$	primitive type
	$\mid t\&P$	tagged type
T	$::= T_1 \to T_2 \mid t$	type
Γ	$::= x_1{:}T_1 \ldots x_n{:}T_n$	type context
Δ	$::= P_1{:}T_1 \ldots P_n{:}T_n$	predicate type context

Fig. 1: Syntax

– $v\&P[e]$ is a *checking expression*, a runtime expression (that cannot be written by the programmer) which denotes an intermediate step in the runtime check that value v satisfies predicate P. That is, if e is neither `true` nor `false`, then we still have to evaluate e to complete the check. The checking expressions $v\&P[\texttt{true}]$ and $v\&P[\texttt{false}]$, instead, denote the final result of the check, being success and failure, respectively.

Accordingly with the intuition given above, we will use the following two shortcuts:

– $v\&P[\texttt{true}]$ will be abbreviated by $v\&P$. This is a *tagged value*, denoting the result of a successful check of the predicate, hence we can rely on the fact that $P\,v$ holds.
– $v\&P[\texttt{false}]$ will be abbreviated by `error` P. This is a dynamic error, tagged by P to denote that there has been a failure in checking P.

Values are abstractions, or values of primitive types, or tagged values as explained above. Note that we take a stratified approach, where only non-functional values can be tagged. Tagged values are identified modulo order and repetitions of tags, e.g., 1&`positive`&`odd` and 1&`odd`&`positive` are the same value.

Operational Semantics The reduction relation $e \to e'$ assumes a given *predicate table* $PT = p_1 \ldots p_n$, which is a sequence of *predicate definitions* which associate to a predicate name a lambda expression (expected to be a predicate, that is,

to return a boolean value). We assume, as usual, that order of definitions is immaterial, and there are no multiple definitions for the same predicate name. That is, the predicate table can be seen as a (partial) map from predicate names to predicates.

Reduction rules are given in Fig.2.

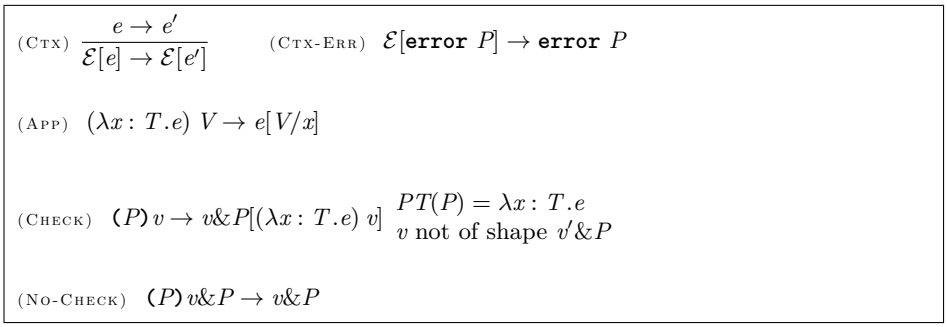


Fig. 2: Reduction rules

The first two rules are standard contextual closure and error propagation rules. Evaluation contexts, given in Fig.1, are as expected. The third rule is (call-by-value) application rule. We denote by $e[e'/x]$ the standard capture-avoiding substitution of occurrences of x in e by e'.

The novel rule (Check) models starting the runtime check that predicate named P holds on value v, by triggering the evaluation of the corresponding predicate application. This rule is applied when v is not tagged by P yet. Otherwise, this means that v is a value obtained through a previous check of predicate named P, hence it is useless to perform the check again, and the value is directly returned, as shown in rule (No-Check). Note that an implementation could keep the tags and follow the same strategy, avoiding useless checks, or be based on type erasure, but in this case checks should be repeated.

Type System Following the stratified approach mentioned above, there are two forms of types:

- non-functional types, called *base types*, which are primitive types possibly tagged with a sequence of predicate names
- functional types, which cannot be tagged.

As for tagged values, we assume that tagged types are identified modulo order and repetitions of tags, e.g., `int`&`positive`&`odd` and `int`&`odd`&`positive` are the same type.

The subtyping relation is given in Fig.3. Rule (Sub-Tag) expresses the expected property that adding tags we get more specific types, other rules are standard.

The typing judgment, $\Gamma \vdash e : T$, says that expression e has type T under the *type context* Γ, where type contexts, Γ, are sequences of assignments of types to

$$(\textsc{Sub-Tag})\ \ t\&P \leq t \qquad (\textsc{Sub-Fun})\ \ \frac{T'_1 \leq T_1 \quad T_2 \leq T'_2}{T_1 \to T_2 \leq T'_1 \to T'_2}$$

$$(\textsc{Sub-Tr})\ \ \frac{T_1 \leq T_2 \quad T_2 \leq T_3}{T_1 \leq T_3} \qquad (\textsc{Sub-Refl})\ \ T \leq T$$

Fig. 3: Subtyping rules

variables, see Fig.1. In Fig.4 we give the rules for deriving the typing judgment. We assume a given *predicate type context* Δ, which is a sequence of assignments of types to predicate names, see Fig.1.

$$(\textsc{T-Var})\ \ \frac{}{\Gamma \vdash x : T}\ \Gamma(x) = T \qquad (\textsc{T-Abs})\ \ \frac{\Gamma, x{:}T \vdash e : T'}{\Gamma \vdash \lambda x : T.e : T \to T'}$$

$$(\textsc{T-App})\ \ \frac{\Gamma \vdash e_1 : T \to T' \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash e_1\ e_2 : T'}\ T_2 \leq T$$

$$(\textsc{T-Check})\ \ \frac{\Gamma \vdash e : t}{\Gamma \vdash (P)\, e : t\&P}\ \begin{array}{l}\Delta(P) = t' \to \texttt{bool} \\ t \leq t'\end{array}$$

$$(\textsc{T-Checking})\ \ \frac{\Gamma \vdash v : t \quad \Gamma \vdash e : \texttt{bool}}{\Gamma \vdash v\&P[e] : t\&P}\ \begin{array}{l}\Delta(P) = t' \to \texttt{bool} \\ t \leq t'\end{array}$$

$$(\textsc{T-Error})\ \ \frac{\Gamma \vdash v : t}{\Gamma \vdash v\&P[\texttt{false}] : T}\ \begin{array}{l}\Delta(P) = t' \to \texttt{bool} \\ t \leq t'\end{array}$$

Fig. 4: Typing rules

The first three rules of Fig.4 are standard rules of the simply-typed lambda calculus. We omit obvious rules for boolean and integer constants.

In rule (T-Check), a check expression has the type of the argument tagged by the predicate name. The argument type t must be a subtype of the parameter type of the predicate. Note that the explicit subtyping condition, rather than an implicit subsumption rule, is necessary to assign to the check expression type $t\&P$, which is possibly more specific than $t'\&P$. For instance, the predicate `positive` requires an `int`, the argument could be an `int&odd`, and we want to get an `int&odd&positive`.

Rule (T-Checking) enforces the restriction that, in a checking expression of type $t\&P$, the value v must have type t, and the expression e must be of type `bool`, so that, if it converges to a value, then the value is either `true` or `false`. As for rule (T-Check), the type of the value must be a subtype of the parameter type of the predicate.

Finally, rule (T-Error) states that an error has any type.[6] Since we encode errors as checking expressions where the expression is `false`[7], we also require for uniformity the same conditions on v of the two previous rules.

Results To prove soundness, expressed as usual by subject reduction and progress theorems, we assume that the predicates in the predicate table be well-typed w.r.t. the predicate type context, that is, for all $P \in \mathtt{dom}(PT)$, $\emptyset \vdash PT(P) : \Delta(P)$. The proof of subject reduction relies on the (standard) substitution and context lemmas, that follow.

Lemma 1 (Substitution). *If $\Gamma, x{:}T' \vdash e : T$, and $\Gamma \vdash e' : T'$, then $\Gamma \vdash e[e'/x] : T$.*

Lemma 2 (Context). *Let $\Gamma \vdash \mathcal{E}[e] : T$, then $\Gamma \vdash e : T'$ for some T', and if $\Gamma \vdash e' : T''$, with $T'' \leq T'$, then $\Gamma \vdash \mathcal{E}[e'] : T$, for all e'.*

Theorem 1 (Subject reduction). *Let e be such that, for some Γ and T, we have that $\Gamma \vdash e : T$. If $e \to e'$, then $\Gamma \vdash e' : T'$ for some T' such that $T' \leq T$.*

Proof. By induction on the rule used for $e \to e'$.

If the rule applied is (Ctx), then we have the thesis by induction hypothesis, using Lemma 2.

If the rule applied is (Ctx-Err), since `error` P is abbreviation for $v\&P[\mathtt{false}]$, from (T-error) of Fig.4 we have that $\Gamma \vdash v\&P[\mathtt{false}] : T$. This proves the thesis.

If the rule applied is (App), then $e = (\lambda x : T_1.e_1)\ V$, $e' = e_1[V/x]$. The proof is standard by using typing rules (T-App), (T-Abs), and Lemma 1.

If the rule applied is (Check), then

1. $e = \mathtt{(}P\mathtt{)}\, v$,
2. $e' = v\&P[(\lambda x : t'.e_p)\ v]$, where $PT(P) = \lambda x : t'.e_p$.

From $\Gamma \vdash \mathtt{(}P\mathtt{)}\, v : T$, and typing rule (T-Check) for some t and t' we have that: $T = t\&P$, $\Gamma \vdash v : t$, $\Delta(P) = t' \to \mathtt{bool}$, and $t \leq t'$. Moreover, we know that $\emptyset \vdash PT(P) : t' \to \mathtt{bool}$. Therefore, also $\Gamma \vdash PT(P) : t' \to \mathtt{bool}$. Applying rule (T-app) of Fig.4, we get that $\Gamma \vdash (\lambda x : t'.e_p)\ V : \mathtt{bool}$. Therefore, from $\Gamma \vdash v : t$, $\Gamma \vdash (\lambda x : t'.e_p)\ V : \mathtt{bool}$, $\Delta(P) = t' \to \mathtt{bool}$, and $t \leq t'$, applying rule (T-checking) of Fig.4, we derive $\Gamma \vdash e' : T$.

If the rule applied is (No-Check), then

1. $e = \mathtt{(}P\mathtt{)}\, v\&P[\mathtt{true}]$, and
2. $e' = v\&P[\mathtt{true}]$.

[6] Recall that this is a standard typing rule in calculi with dynamic errors, needed to ensure that error propagation, see rule (Ctx-Err), preserves type.

[7] We could have, alternatively, one more reduction step into an additional error expression, but we prefer this more succinct formulation.

From $\Gamma \vdash e : T$, and typing rule (T-Check) for some t and t' and we have that: $T = t\&P$, $\Gamma \vdash v\&P[\texttt{true}] : t$, $\Delta(P) = t' \to \texttt{bool}$, and $t \leq t'$. From $\Gamma \vdash v\&P[\texttt{true}] : t$, and typing rule (T-Checking) we have that t is already of shape $t''\&P$, for some t''. Therefore, $t = T$, and $\Gamma \vdash e' : t$. □

The proof of progress relies on the canonical forms lemma, which characterizes the shape of values of specific types.

Lemma 3 (Canonical forms).

1. *If* $\vdash V : T_1 \to T_2$, *then* $V = \lambda x : T_1.e$.
2. *If* $\vdash V : \texttt{bool}$, *then either* $V = \texttt{false}$, *or* $V = \texttt{true}$.
3. *If* $\vdash V : \texttt{int}$, *then* $V = n$.
4. *If* $\vdash V : t\&P$, *then* $V = v\&P$, *i.e.,* $V = v\&P[\texttt{true}]$.

Theorem 2 (Progress). *Let e be such that, for some T, we have that $\emptyset \vdash e : T$. Then, either $e = V$, for some V, or $e = \texttt{error}\ P$, or $e \to e'$ for some e'.*

Proof. If $e \neq V$, for some V, and $e \neq \texttt{error}\ P$, then either $e = e_1\, e_2$, or $e = \texttt{(}P\texttt{)}\, e_1$, or $e = v\&P[e_1]$. We consider the last two cases.

If $e = \texttt{(}P\texttt{)}\, e_1$, and e_1 is not a value or error, then by induction hypothesis $e_1 \to e_2$. Therefore, $\texttt{(}P\texttt{)}\, e_1 \to \texttt{(}P\texttt{)}\, e_2$ with (Ctx) and $\mathcal{E} = \texttt{(}P\texttt{)}[\,]$.
If $e_1 = \texttt{error}\ P$, then $e \to \texttt{error}\ P$ with (Ctx-Err) and $\mathcal{E} = \texttt{(}P\texttt{)}[\,]$.
If $e_1 = V$, for some V, from $\emptyset \vdash \texttt{(}P\texttt{)}\, V : T$, and typing rule (T-Check), we have that, for some t and t': $T = t\&P$, $\emptyset \vdash V : t$, $\Delta(P) = t' \to \texttt{bool}$, and $t \leq t'$.
There are two cases for type t: either $t = t'\&P$ for some t', that is, the type is already tagged with P, or not.
If $t = t'\&P$ for some t', then, from Lemma 3.4, $V = v\&P$, and rule (no-check) is applicable. So $e \to v\&P$.
Otherwise rule (check) is applicable, and $e \to V\&P[(\lambda x : t'.e_p)\ V]$.

If $e = v\&P[e_1]$, and e_1 is not a value, as in the previous case we can apply either rule (Ctx) or (Ctx-Err).
If $e_1 = V$, for some V, then, from $\emptyset \vdash v\&P[e_1] : T$, and typing rule (T-Checking), we have that $\emptyset \vdash e_1 : \texttt{bool}$. Therefore, from Lemma 3.2, $e_1 = \texttt{true}$ or $e_1 = \texttt{false}$. If $e_1 = \texttt{true}$, then e is the value $v\&P$, and if $e_1 = \texttt{false}$, then $e = \texttt{error}\ P$. □

In addition to soundness, we expect the following key property to hold: if an expression of a tagged type $t\&P$ reduces to a value, then this value satisfies predicate P. This is implied by subject reduction, plus the fact that a value v of a tagged type $t\&P$ satisfies predicate P. The latter property is true "by construction" in our calculus, provided that we consider as values only the subset of syntactic values which can be actually obtained by reduction. (Recall that checking expressions cannot be written by the programmer.) This is formalized by the notion of *admissible expression* below.

Definition 1. *The expression e is* admissible *if, for all subexpressions $v\&P[e']$ of e, we have $(\lambda x : T.e_p)\ v \to^\star e'$, where $PT(P) = \lambda x : T.e_p$.*

Note that expressions written by the programmer are always admissible since they do not have subexpressions which are checking expressions.

Lemma 4. *If e is admissible, and $e \to e'$, then e' is admissible.*

Proof. By induction on the reduction rules. The interesting case is:
(Check) $(P)\, v \to v\&P[(\lambda x : T.e)\ v]\ PT(P) = \lambda x : T.e$
In this case, the thesis holds since $(\lambda x : T.e)\ v$ reduces to itself in zero steps. □

Theorem 3. *Let e be an admissible expression such that, for some Γ and T, we have that $\Gamma \vdash e : T$. If $e \to^\star v\&P$, then $(\lambda x : T.e_p)\ v \to^\star$ `true`, where $PT(P) = \lambda x : T.e_p$.*

Proof. By subject reduction (Theorem 1) and Lemma 4. □

3 Related work

Constrained types and type invariants The work most closely related to ours are likely *constrained types* of X10 [10]. A constrained type `C{c}` consists of a class or interface `C` and a constraint `c` on the immutable state of `C` and in-scope final variables. The system is parametric on the underlying constraint system: the compiler implements simple equality-based constraints but, in addition, supports extension with new constraint systems using compiler plugins. For instance, the constraint solver automatically detects subtyping in cases such as `Int[self>5]` $\leq$ `Int[self>0]`. Their main goal is to support static verification as much as possible, by allowing in constraints a very restricted subset of the language.[8] On the other hand, in our approach the programmer can express *any* property in the predicate language, which is Turing complete being the language itself. This could be coupled with static verification for conditions expressed in a limited sub-language, but we do not require all the predicates to be in such category.

Whiley [11] offers *type invariants*, conceptually similar to constrained types but, as in our approach, they use the full language in predicates. Whiley is designed from scratch, with the aim of permitting both dynamic (runtime) verification and static one. As X10, Whiley attempts at automatically inferring predicate subtyping. In a personal communication with David Pearce (Whiley project leader) he agreed that static verification of predicates and subtyping inference are orthogonal with respect to the main idea of checking a property and then propagating such knowledge, and that a core model of such idea would be very valuable.

Another important difference is that, in both X10 and Whiley, interpretation of predicates is *structural*, that is, the meaning of a predicate is its definition. Our interpretation is *nominal* instead, that is, , the meaning of a predicate is its name, and the current definition is just an implementation of such concept. Under our lens, predicate subtyping must be defined explicitly by the programmer, if needed, see Sect.4.

[8] The user is able to escape the confines of static type-checking using dynamic casts, as in our approach.

Pre/post conditions In the style of runtime verification, a method can have *preconditions* on its arguments that are dynamically checked when the method is invoked. If the preconditions do not hold, the caller is blamed. Then, before producing the result, the *postcondition* is verified. If the postcondition does not hold, the method implementation is blamed, see, e.g., [8].

In our approach, a method can encode preconditions as types of its parameters, guaranteeing that the check is performed at the client side.[9] Then, the method may cast the result to a more specific type, as a way to check the postcondition. An advantage of encoding a precondition as a type is that in this way the information that a value satisfies the precondition can be propagated to internal calls with the same precondition, rather than be checked again.

Runtime certification Our work can be seen as a (circular) variation over *runtime certification* as in Athena [2]. In a nutshell, a program is runtime certified if, when results are produced, they are guaranteed to respect certain properties. Runtime certification is usually proposed together with a specific proof language, different from the computation language. Our approach instead uses a single language, and the correctness of a value is defined as the satisfaction of certain predicates written in the language itself.

Refinement types and blame calculi *Refinement types*, introduced in [4], are a system of subtypes for a polymorphically typed language like ML, which are used to specify properties of user-defined data types. The properties are expressed as union and intersection of types derived from user defined-types, and are statically verified by a decidable type inference algorithm. In [7] are defined two *hybrid calculi*, λ^C and λ^H, whose types include refinement types describing subsets of base types satisfying predicates. Predicates are, as in our calculus, expressions of the language evaluating to boolean values. Expressions include a cast, like our check, which involves checking dynamically that a predicate holds. As for X10 and Whiley, the interpretation of predicates is structural.

The *blame calculus*, see [12], provides a uniform view of static and dynamic type checking. Programmers may add casts to evolve statically typed code into code including refinement types (as in [7], predicates are structural). The blame calculus is more general than our calculus, since refinement is not limited to base types, and, in [1], it is extended to a polymorphically typed lambda calculus. The type system is tailored to detect where the cause of the violation comes from, and, in particular, it is proved that it may not come from statically typed code. We claim that, even though our framework is more limited, since we do not allow properties of functional types, its simplicity, and the nominal approach to refinement, make it easier to implement and integrate in existing languages, see next section.

[9] Preconditions involving more than one parameter could be encoded by using tuple types.

4 Conclusion and future work

We have proposed and formalized a *lightweight* approach for handling values which are required to satisfy a property.

Compared to works discussed in the previous section, which generally rely on separate assertion language, structural interpretation of predicates, and (some) static verification, our design is less powerful, but easier to integrate and implement in existing languages.

In particular, we adopt a meta-circular approach, where the property is implemented by the programmer in the language itself. Note that, in this way, checking that a value has a given property, say P, could lead to divergence. However, the fact that an expression is guaranteed to succeed the corresponding runtime check is a *static* property which can be controlled by the type system. More precisely, an expression of type $t\&P$ can either reduce to a value of the same type, which is guaranteed to succeed the check, or lead to a dynamic error, or not terminate, as it happens for any other type.

Note also that by the nominal interpretation of predicates we gain the well-known advantages of the nominal approach for software evolution: the meaning of a predicate can change, and a system with structural approach would be fragile.

Our design is in principle language independent. To integrate and implement check expressions on top of an existing language, they could be translated, by a pre-processing step, into the application of the corresponding predicate, a function written in the source language. In an imperative language, analogously to what is required in other proposals, predicates should be *pure* functions, and an object could be ensured to invariantly satisfy a property only under some conditions, the simplest being that the object is deeply immutable, that is, all its fields are (recursively) immutable. More permissive conditions could be allowed by combining the type system with modifiers for aliasing and immutability control, see, e.g., [6,5]. Interaction with polymorphic types is an interesting topic for future work. Instantiating polymorphic types with tagged types seems to pose no problems. Moreover, at a first sight, polymorphic types with tags seem to make sense in combination with subtyping constraints, e.g., a predicate `isempty` which holds when the argument, assumed to be of a subtype of `Collection`, is empty.

As future work, we also discuss two extensions.

Predicate subtyping Certain predicates may be stronger than others, for example we expect `greaterThan5` to imply `positive`. Since these predicates are defined as code, there is no general way to discover such implications automatically. However, we could offer a language construct for subtyping relations declared by the programmer, as usual for nominal types, for example, using again a Java-like syntax:

```
bool positive(int x) {return x>=0;}
bool greaterThan5(int x) {return x>5;}
  implies positive
```

This extension could be encoded in the original language by expanding occurrences of the stronger predicate name to a sequence, for instance:

```
int&greaterThan5 x=(greaterThan5)34
```

would be expanded to

```
int&greaterThan5&positive x=(greaterThan5)(positive)34
```

Refinement for pre-existing operations One limitation of the proposed approach is that existing operations are unaware of newly introduced properties.

For example, assume to have a `Sum` operation encoding the sum of two numbers, with type `int*int->int`. If we add the concept of `positive`, we know that, in a context where `a` and `b` are positive, `Sum(a,b)` should be positive; but in the current system it is just an `int`. An extension of our approach could allow:

```
SumPosPos refines Sum:
  int&positive*int&positive->int&positive
```

This extension could be encoded in the original language by declaring an auxiliary function `SumPosPos` which just calls `Sum` and casts the result to `int&positive`. In the case of an overpermissive refinement, the error can be traced back to the declaration of `SumPosPos`.

Acknowledgments. We are grateful to the anonymous reviewers for their useful suggestions, which led to substantial improvements.

References

1. A. Ahmed, R. B. Findler, J. G. Siek, and P. Wadler. Blame for all. In T. Ball and M. Sagiv, editors, *ACM Symp. on Principles of Programming Languages 2011*, pages 201–214. ACM Press, 2011.
2. K. Arkoudas and M. C. Rinard. Deductive runtime certification. *Electronic Notes in Theoretical Computer Science*, 113:45–63, 2005. Fourth Workshop on Runtime Verification (RV 2004).
3. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In *CASSIS'04 - Construction and Analysis of Safe, Secure and Interoperable Smart Devices*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2005.
4. T. Freeman and F. Pfenning. Refinement types for ML. In *ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, pages 268–277. ACM, 1991.
5. P. Giannini, M. Servetto, and E. Zucca. Types for immutability and aliasing control (extended abstract). In *ICTCS'16 - Italian Conf. on Theoretical Computer Science*, 2016. In this volume.
6. C. S. Gordon, M. J. Parkinson, J. Parsons, A. Bromfield, and J. Duffy. Uniqueness and reference immutability for safe parallelism. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2012)*, pages 21–40. ACM Press, 2012.

7. J. Gronski and C. Flanagan. Unifying hybrid types and contracts. In M. T. Morazán, editor, *TFP 2007 - Trends in Functional Programming*, volume 8 of *Trends in Functional Programming*, pages 54–70. Intellect, 2007.
8. G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D. R. Cok. How the design of JML accommodates both runtime assertion checking and formal verification. *Science of Computer Programming*, 55(1-3):185–208, 2005.
9. P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In B. Jobstmann and K. R. M. Leino, editors, *VMCAI'16 - Verification, Model Checking, and Abstract Interpretation*, volume 9583 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2016.
10. N. Nystrom, V. A. Saraswat, J. Palsberg, and C. Grothoff. Constrained types for object-oriented languages. In G. E. Harris, editor, *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2008)*, pages 457–474. ACM Press, 2008.
11. D. J. Pearce and L. Groves. Designing a verifying compiler: Lessons learned from developing Whiley. *Science of Computer Programming*, 113:191 – 220, 2015.
12. P. Wadler and R. B. Findler. Well-typed programs can't be blamed. In G. Castagna, editor, *ESOP 2009 - European Symposium on Programming*, volume 5502 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2009.

On the bisimulation hierarchy of state-to-function transition systems

Marino Miculan and Marco Peressotti

Dept. of Mathematics, Computer Science and Physics, University of Udine, Italy
marino.miculan@uniud.it marco.peressotti@uniud.it

Abstract Weighted labelled transition systems (WLTSs) are an established (meta-)model aiming to provide general results and tools for a wide range of systems such as non-deterministic, stochastic, and probabilistic systems. In order to encompass processes combining several quantitative aspects, extensions of the WLTS framework have been further proposed, *state-to-function transition systems* (FuTSs) and *uniform labelled transition systems* (ULTraSs) being two prominent examples. In this paper we show that this hierarchy of meta-models collapses when studied under the lens of bisimulation-coherent encodings.

1 Introduction

Weighted labelled transition systems (WLTSs) [10] is a meta-model for systems with quantitative aspects: transitions $P \xrightarrow{a,w} Q$ are labelled with *weights* w, taken from a given monoidal weight structure. Many computational aspects can be captured just by changing the underlying weight structure: weights can model probabilities, resource costs, stochastic rates, *etc.*; as such, WLTSs are a generalisation of labelled transition systems (LTSs), probabilistic systems (PLTSs) [6], stochastic systems [9], among others. Definitions and results developed in this setting instantiate to existing models, thus recovering known results and discovering new ones. In particular, the notion of *weighted bisimulation* [10] in WLTSs coincides with strong bisimulation for all the aforementioned models.

In the wake of these encouraging results, other meta-models have been proposed aiming to cover an even wider range of computational models and concepts. *Uniform labelled transition systems* (ULTraSs) [2] are systems whose transitions have the form $P \xrightarrow{a} \phi$, where ϕ is a *weight function* assigning weights to states; hence, ULTraSs can be seen both as a non-deterministic extension of WLTSs and as a generalisation of Segala's probabilistic systems [18] (NPLTSs). In [14, 15] a (coalgebraically derived) notion of bisimulation for ULTraSs is presented and shown to precisely capture bisimulations for weighted and Segala

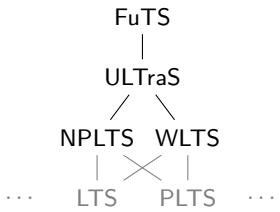
V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 88–102 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

systems. *Function-to-state transition systems* (FuTSs) were introduced in [5] as a generalisation of the above, of IMC [8], and other models. Later, [11] defines a (coalgebraically derived) notion of bisimulation for FuTSs which instantiates to known bisimulations for the aforementioned models.

Given all these meta-models, it is natural to wonder about their *expressiveness*. We should consider not only the class of systems these frameworks can represent, but also *whether* these representations are faithful with respect to the properties we are interested in. Intuitively, a meta-model M is *subsumed by* M' *according to a property* P if any system S which is an instance of M with the property P, is also an instance of M' preserving P.

In this paper, we aim to classify meta-models according their ability to correctly express *strong bisimulation*. Therefore, in our contest a meta-model M is *subsumed* by M' if any system S which is an instance of M, is also an instance of M' preserving strong bisimulation.

Previous work [2, 10, 11, 14, 15] have shown that, according to this order, each of the meta-models mentioned above subsumes the previous ones, thus forming the hierarchy shown aside. Still, an important question is open: is any of these meta-models *strictly more expressive* than others? In this work we address this question, proving that this is not the case: the black part of the hierarchy collapses!



In order to formally capture the notion of "expressiveness order between system classes with respect to (strong) bisimulation", we introduce the notion of *reduction* between classes of systems. Although the driving motivation is the study of the FuTSs hierarchy under the lens of bisimulation, the notion of reduction is more general and, as defined in this work, can be used to study any class of state-based transition systems. In fact, all the constructions and results are developed abstracting from the "shape" of computation under scrutiny.

Synopsis Section 2 recalls an abstract and uniform account of transition systems on discrete state spaces, akin to [17]. Section 3 presents a general construction for extending equivalence relations over sets of states to sets of behaviours. Building on this relational extension, Section 4 provides a characterisation of (strong) bisimulations in a modular fashion. The notion of reduction is introduced in Section 5, along with general reductions. In Section 6 we provide a reduction from the category of FuTSs to the category of WLTSs together with intermediate reductions for special cases of FuTSs such as ULTraSs, nested FuTSs, and combined FUTSs. Final remarks are in Section 7 and omitted proofs in Appendix A.

2 Discrete transition systems

For an alphabet A and set of states X, the function space X^A is understood as the set of all possible behaviours characterising deterministic input over A. In this context, a transition system exposing this computational behaviour is precisely described by a function $\alpha\colon X \to X^A$ mapping each state $x \in X$ to

some element in X^A. For a function $f\colon X \to Y$ and $\phi \in X^A$, the assignment $\phi \mapsto f \circ \phi$ defines a function $(f)^A\colon X^A \to Y^A$ that extends the action of f from state spaces X and Y to behaviours defined over them in a coherent way. A function $f\colon X \to Y$ between the state spaces of systems (say, $\alpha\colon X \to X^A$ and $\beta\colon Y \to Y^A$) preserves and reflect their structure whenever $f^A \circ \alpha = \beta \circ f$. Since they preserve and reflect the transition structure of systems, these functions are called *homomorphisms* (which are functional bisimulations, *cf.* [17, Thm. 2.5]).

All the structures and observations described in the above example stem from a single information: the "type" of the behaviour under scrutiny. This is well understood as an *endofunctor* over the category of state spaces [17] — in this context, the category of sets and functions.

Non-deterministic transitions are captured by the powerset endofunctor $\mathcal{P}$ mapping each set X its powerset $\mathcal{P}X$ and function f to its inverse image $\mathcal{P}f$ *i.e.* the function given by the assignment $Z \mapsto \{f(z) \mid z \in Z\}$. Since subsets are functions weighting elements over the monoid $\mathbb{B} = (\{\mathtt{tt}, \mathtt{ff}\}, \vee, \mathtt{ff})$, the above readily extends to quantitative aspects (such as probability distributions, stochastic rates, delays, etc.) by simply considering other a non-trivial abelian monoids[1] [10,12,15]. This yields the endofunctor $\mathcal{F}_M$ which assigns

- to each set X the set $\{\phi\colon X \to M \mid |\{x \mid \phi(x) \neq 0\}| \in \mathbb{N}\}$;
- to each function $f\colon X \to Y$ the map $(\mathcal{F}_M f)(\phi) = \lambda y \in Y. \sum_{x:f(x)=y} \phi(x)$. (summation is well defined because ϕ is finitely supported, by above).

Probabilistic computations are a special case of the above where weight functions are distributions (*cf.* [10]) and are captured by the endofunctor $\mathcal{D}$ given on each set X as $\mathcal{D}X = \{\phi \in \mathcal{F}_{[0,\infty)}X \mid \sum \phi(x) = 1\}$ and on each function f as $\mathcal{F}_{[0,\infty)}f$.

From this perspective, $\mathcal{D}$ can be thought as a sort of "subtype" of $\mathcal{F}_{[0,\infty)}$. This situation is formalised by means of (component-wise) injective natural transformations (herein *injective transformations*). Composition and products of natural transformations are component-wise and the class of injective ones is closed under such operations. In general, for an injective transformation μ and a n endofunctor T, μ_T is again injective but $T\mu$ may not be so. The latter is injective given that T *preserves injective maps i.e.* Tf is injective whenever f is injective. All the examples listed in this paper meet this mild assumption.

Lemma 1. *Any composition and product of Id, $\mathcal{P}$, $\mathcal{F}_M$ preserve injections.*

Example 1. The endofunctor $\mathcal{P}\mathcal{F}_M$ models the alternation of non-deterministic steps with quantitative aspects captured by $(M, +, 0)$. There is an injective transformation $\eta\colon Id \to \mathcal{P}$ whose components are given by the mapping $x \mapsto \{x\}$ and hence, by composition, $\eta_{\mathcal{F}_M}\colon \mathcal{F}_M \to \mathcal{P}\mathcal{F}_M$ is an injective transformation. □

Definition 1. *For an endofunctor T over* Set*, a transition system of type T (*T-system*) is a pair (X, α) where X is the set of states (*carrier*) and $\alpha\colon X \to TX$ is the* transition map*. For (X, α) and (Y, β) T-systems, a* T-homomorphism *from the former to the latter is a function $f\colon X \to Y$ s.t. $Tf \circ \alpha = f \circ \beta$.*

[1] An abelian monoid is a set M equipped with an associative and commutative binary operation $+$ and a unit 0 for $+$; such structure is called trivial when M is a singleton.

Since system homomorphism composition is defined in terms of composition of the underlying functions on carriers it is immediate to check that the operation is associative and has identities. Therefore, any class of systems together with their homomorphisms defines a category.

We adopt the following notational conventions. A transition system (X, α) is referred by its transition map only; in this case its carrier is written $\mathrm{car}(\alpha)$. Homomorphisms are denoted by their underlying function. Categories of systems are written using sans serif font with $\mathsf{Sys}(T)$ being the category of all T-systems and T-homomorphisms and $\mathsf{C}|_T$ its subcategory of systems in the category C.

Example 2 (LTSs). For a set A of labels, labelled transition systems are $(\mathcal{P}-)^A$-systems, and image finite LTSs $(\mathcal{P}_f-)^A$-systems [17]. Hereafter let LTS denote the category of all image-finite labelled transition systems and let $\mathsf{LTS}(A) \triangleq \mathsf{Sys}((\mathcal{P}_f-)^A)$ be its subcategory of systems labelled over A. □

Example 3 (WLTSs). For a set of labels A and an abelian monoid M, weighted labelled transition systems are characterised by the endofunctor $(\mathcal{F}_M-)^A$ [10] and hence form the category $\mathsf{WLTS}(A, M) \triangleq \mathsf{Sys}((\mathcal{F}_M-)^A)$ *i.e.* the (A, M)-indexed component of WLTS, the category of all WLTSs. When the monoid $\mathbb{B}$ of boolean values under disjunction is considered, $\mathsf{WLTS}(A, \mathbb{B})$ is $\mathsf{LTS}(A)$. □

Example 4 (ULTraSs). We adopt the presentation of ULTraSs given in [14, 15]. For a set of labels A and an abelian monoid M, uniform labelled transition systems are characterised by the endofunctor $(\mathcal{P}\mathcal{F}_M-)^A$; image finite ULTraSs by $(\mathcal{P}_f\mathcal{F}_M-)^A$. We denote by ULTraS the category of all image-finite ULTraSs and by $\mathsf{ULTraS}(A, M)$ its subcategory of systems with labels in A and weights in M. WLTSs can be cast to ULTraSs by means of the injective transformation $(\eta_{\mathcal{F}_M})^A$ described in Example 1. These ULTraSs are called in [2] *functional*. □

Example 5 (FuTSs). FuTSs are T-systems for T generated by the grammar

$$T ::= (S-)^A \mid T \times (S-)^A \qquad S ::= \mathcal{F}_M \mid \mathcal{F}_M \circ S$$

where A and M range over (non-empty) sets of labels and (non-trivial) abelian monoids, respectively. Any such endofunctor is equivalently described by:

$$(\mathcal{F}_{\vec{M}} f)^{\vec{A}} \triangleq \prod_{i=0}^{n} (\mathcal{F}_{\vec{M}_i} f)^{A_i} \qquad \text{and} \qquad (\mathcal{F}_{\vec{M}_i} f)^{A_i} \triangleq (\mathcal{F}_{M_{i,0}} \dots \mathcal{F}_{M_{i,m_i}} f)^{A_i}$$

for $\vec{A} = \langle A_0, \dots, A_n \rangle$ a sequence of non-empty sets, $\vec{M}_i = \langle M_{i,0}, \dots, M_{i,m_i} \rangle$ a sequence of non-trivial abelian monoids, and $\vec{M} = \langle \vec{M}_0, \dots, \vec{M}_n \rangle$ [12, 15]. For any $\vec{A}$ and $\vec{M}$ as above define $\mathsf{FuTS}(\vec{A}, \vec{M})$ as $\mathsf{Sys}((\mathcal{F}_{\vec{M}}-)^{\vec{A}})$. Clearly, $\mathsf{FuTS}(\langle A \rangle, \langle M \rangle)$ and $\mathsf{FuTS}(\langle A \rangle, \langle B, M \rangle)$ coincide with $\mathsf{WLTS}(A, M)$ and $\mathsf{ULTraS}(A, M)$, respectively. Then, LTS, WLTS, and ULTraS are subcategories of FuTS, the category of all FuTSs. For $\vec{M} = \langle \langle M_{0,0}, \dots, M_{0,m_0} \rangle, \dots, \langle M_{n,0} \dots M_{n,m_n} \rangle \rangle$ as above, recall from [12] that a FuTS over $\vec{M}$ is called: *nested* if $n = 0$, *combined* if $m_i = 0$ for each $i \in \{0, \dots, n\}$, and *simple* if it is both combined and nested. □

3 Equivalence extensions

Several definitions of bisimulation found in literature use (more or less explicitly) some sort of extension of equivalence relations from state spaces to behaviours over these spaces. For instance, in [18] two probability distributions are considered equivalent with respect to an equivalence relation R on their domain if they assign the same probability to any equivalence class induced by R:

$$\phi \equiv_R \psi \overset{\triangle}{\iff} \forall C \in X/R \ \left(\textstyle\sum_{x\in C} \phi(x) = \sum_{x\in C} \psi(x)\right).$$

This section defines equivalence extensions for arbitrary endofunctors (over Set) and studies how constructs such as composition or products reflect on these extensions, providing some degree of modularity.

Definition 2. *For an equivalence relation R on X its* T-extension *is the equivalence relation R^T on TX such that $\phi\ R^T\ \psi \overset{\triangle}{\iff} (T\kappa)(\phi) = (T\kappa)(\psi)$ where $\kappa\colon X \to X/R$ is the canonical projection to the quotient induced by R.*

As an example, let us consider the endofunctor $(-)^A$ describing deterministic inputs on A: the resulting extension for an equivalence relation R relates functions mapping the same inputs to states related by R. Formally:

$$\phi\ R^{(-)^A}\ \psi \iff \kappa \circ \phi = \kappa \circ \psi \iff \forall a \in A\ (\phi(a)\ R\ \psi(a)).$$

Extensions for $\mathcal{P}$ are precisely "subset closure" of relations (*cf.* [15]) and relate all and only those subsets for which the given relation is a correspondence. Formally:

$$\begin{aligned} Y\ R^{\mathcal{P}}\ Z &\iff \{\kappa(y) \mid y \in Y\} = \{\kappa(z) \mid z \in Z\} \\ &\iff (\forall y \in Y \exists z \in Z (y\ R\ z)) \wedge (\forall z \in Z \exists y \in Y (y\ R\ z)) \end{aligned}$$

Extension for $\mathcal{F}_M$ are generalise the subset closure to multisets and relate only weight functions assigning the same cumulative weight to each equivalence class induced by R: $\phi\ R^{\mathcal{F}_M}\ \psi \iff \forall C \in X/R \left(\sum_{x\in C} \phi(x) = \sum_{x\in C} \psi(x)\right)$ In particular, $R^{\mathcal{D}}$ is precisely Segala's equivalence $\equiv_R$ [18].

Consider extensions for the endofunctor $(\mathcal{P}-)^A$ describing LTSs:

$$\phi\ R^{(\mathcal{P}-)^A}\ \psi \iff \forall a \in A \begin{pmatrix} (\forall y \in \phi(a)\, \exists z \in \psi(a)\, (y\ R\ z)) \wedge \\ (\forall z \in \psi(a)\, \exists y \in \phi(a)\, (y\ R\ z)) \end{pmatrix}$$

Clearly, $R^{\mathcal{P}(-)^A}$ can be equivalently written as

$$\phi\ R^{\mathcal{P}(-)^A}\ \psi \iff \forall a \in A (\phi(a)\ R^{\mathcal{P}}\ \psi(a))$$

which suggests some degree of modularity in the definition of extensions to composite endofunctors. In general, this kind of reformulations is not possible since for arbitrary endofunctors T ans S, it holds only that $\phi\ \left(R^S\right)^T\ \psi \implies \phi\ R^{T\circ S}\ \psi$. The converse implication holds whenever T preserves injections.

Lemma 2. *$\left(R^S\right)^T \subseteq R^{T\circ S}$ and $\left(R^S\right)^T \supseteq R^{T\circ S}$, given T preserves injections.*

Endofunctors modelling inputs, such as $(-)^A$ and $(\mathcal{P}_f -)^A$, can be seen as products (in these cases as powers) of endofunctors indexed over the input space A. As suggested by the above examples, for product endofunctors it holds that:

$$\phi\ R^{(\prod T_i)}\ \psi \iff \forall i \in I(\pi_i(\phi)\ R^{T_i}\ \pi_i(\psi))$$

where $\pi_i \colon \prod T_i X \to T_i X$ is the projection on the i-th component of the product.

Lemma 3. *For $I \neq \emptyset$ and $\{T_i\}_{i\in I}$, $R^{\left(\prod_{i\in I} T_i\right)} \cong \prod_{i\in I} R^{T_i}$.*

FuTSs offer an instance of the above result: the endofunctor $(\mathcal{F}_{\vec{M}} -)^{\vec{A}}$ modelling FuTSs over $\vec{M} = \langle \vec{M}_0, \ldots, \vec{M}_n \rangle$ and $\vec{A} = \langle A_0; \ldots; A_n \rangle$ is a product indexed over $\{(i, a) \mid i \leq n \wedge a \in A_i\}$. Thus, the extension $R^{(\mathcal{F}_{\vec{M}} -)^{\vec{A}}}$ is described by:

$$\phi\ R^{(\mathcal{F}_{\vec{M}} -)^{\vec{A}}}\ \psi \iff \forall i \leq n \forall a \in A_i(\phi_i(a)\ R^{\mathcal{F}_{\vec{M}_i}}\ \psi_i(a)).$$

For an equivalence relation R define its restriction to X as the equivalence relation $R|_X \triangleq R \cap (X \times X)$. Both $(R|_X)^T$ and $R^T|_{TX}$ are equivalence relations over the set of T-behaviours for X and, in general, the former is finer than the latter, unless T preserves injections—in such case, the two coincide.

Lemma 4. *For R and equivalence relation on Y and $X \subseteq Y$, $(R|_X)^T \subseteq R^T|_{TX}$, and, provided T preserves injections, $(R|_X)^T \supseteq R^T|_{TX}$.*

Intuitively, this result allows us to encode multiple steps sharing the same computational aspects as single steps at the expense of bigger state spaces. In fact, it follows that $(R|_X)^{T^{n+1}} = R^T|_{T^n X}$, assuming T preserves injections.

Lemma 5. *Let $\mu \colon T \to S$ be an injective natural transformation. For R an equivalence relation on X, $\phi\ R^T\ \psi \iff \mu_X(\phi)\ R^S\ \mu_X(\psi)$.*

4 Bisimulations

In this section we give a general definition of bisimulation based on the notion of equivalence relation extension introduced above. This approach is somehow modular, as the definition reflects the structure of the endofunctors characterising systems under scrutiny. This allows to extend results developed in Section 3 to bisimulation and, in Section 5, to reductions.

Definition 3. *An equivalence relation R is a* strong T-bisimulation *(herein, bisimulation) for a T-system α iff $x\ R\ x' \implies \alpha(x)\ R^T\ \alpha(x')$. We denote by* $\mathrm{bis}(\alpha)$ *the set of all bisimulations for the system α.*

The notion of bisimulations as per Definition 3 coincides with Aczel-Mendler's notion of *precongruence* [1].

Definition 4. *An equivalence relation R on X is a (Aczel-Mendler) precongruence for $\alpha\colon X \to TX$ iff, for any two functions $f, f'\colon X \to Y$ such that $x\ R\ x' \implies f(x) = f'(x')$ it holds that $x\ R\ x' \implies (Tf \circ \alpha)(x) = (Tf' \circ \alpha)(x')$.*

Theorem 1. *For α a T-system, every strong T-bisimulation for α is an AM-precongruence and* vice versa*.*

Bisimulations for systems considered in this paper are known to be *kernel bisimulations* (*cf.* [10,12,15,17]) *i.e.* kernels of functions carrying homomorphisms from systems under scrutiny [19]. These can be intuitively thought as defining *refinement systems* over the equivalence classes they induce.

Definition 5. *A relation R on X is a kernel bisimulation for $\alpha\colon X \to TX$ iff there is $\beta\colon Y \to TY$ and $f\colon \alpha \to \beta$ s.t. R is the kernel of the map underlying f.*

In general, Definition 3 is stricter than Definition 5 but the two coincide for endofunctors preserving (enough) injections—*e.g.* any example from this paper.

Corollary 1. *For $\alpha\colon X \to TX$, the following are true:*
- *A bisimulation for α is a kernel bisimulation for α.*
- *If T preserves injections, a kernel bisimulation for α is a bisimulation for α.*

From Corollary 1 and Lemma 1 it follows that Definition 3 captures strong bisimulation for LTSs [16], for WLTSs [10], for Segala systems [18], for ULTraSs [15], and for FuTSs [12], since these are all instances of kernel bisimulations.

Lemma 6. *For $T = \prod_{i\in I} T_i$ and $\alpha \in \mathsf{Sys}(T)$, $\mathrm{bis}(\alpha) = \bigcap_{i\in I} \mathrm{bis}(\pi_i \circ \alpha)$.*

A special but well known instance of Lemma 6 is given by definitions of bisimulations found in the literature for LTSs, WLTSs and in general FuTSs. In fact, all these bisimulation contain a universal quantification over the set of labels. For instance, a R is a bisimulation for an LTS $\alpha\colon X \to (\mathcal{P}X)^A$ iff:

$$x\ R\ x' \implies \forall a \in A \left(\begin{array}{l} (\forall y \in \phi(a)\, \exists z \in \psi(a)\, (y\ R\ z)) \wedge \\ (\forall z \in \psi(a)\, \exists y \in \phi(a)\, (y\ R\ z)) \end{array} \right)$$

that is, iff R is the intersection of an A-indexed family composed by a bisimulation for each transition system $\alpha_a\colon X \to \mathcal{P}X$ projection of α on $a \in A$.

Lemma 7. *For $n \in \mathbb{N}$ and $\alpha \in \mathsf{Sys}(T^{n+1})$, there is $\underline{\alpha} \in \mathsf{Sys}(T)$ such that:*
- *$R \in \mathrm{bis}(\alpha) \implies \exists R' \in \mathrm{bis}(\underline{\alpha})(R = R'|_{\mathrm{car}(\alpha)})$,*
- *$R \in \mathrm{bis}(\underline{\alpha}) \implies R|_{\mathrm{car}(\alpha)} \in \mathrm{bis}(\alpha)$.*

Proof. Let $\underline{X}$ be $\coprod_{i=0}^{n} T^i X$ and $\underline{\alpha}\colon \underline{X} \to T(\underline{X})$ be $[T\iota_n\alpha_0, T\iota_0\alpha_1 \ldots, T\iota_{n-2}\alpha_{n-1}]$ where $\iota_i\colon T^i X \to \underline{X}$ is the i-th coproduct injection, $\alpha_0\colon X \to T(T^n X)$ is α, and $\alpha_{i+1}\colon T^{i+1}X \to T(T^i X)$ is given by the identity for $T^{i+1}X$. If $R \in \mathrm{bis}(\overline{\alpha})$, then:

$$\begin{aligned} x\ R|_X\ x' &\implies x\ R\ x' \overset{(i)}{\implies} \underline{\alpha}(x)\ R^T\ \underline{\alpha}(x') \overset{(ii)}{\iff} \alpha(x)\ R^T|_{T^{n+1}X}\ \alpha(x') \\ &\overset{(iii)}{\iff} \alpha(x)\ (R|_X)^{T^{n+1}}\ \alpha(x') \end{aligned}$$

where (i) follows by $R \in \mathrm{bis}(\underline{\alpha})$; (ii) follows by noting that $\underline{\alpha}$ acts as α on X and hence both $\underline{\alpha}(x) = \alpha(x)$ and $\underline{\alpha}(y) = \alpha(y)$ are elements of $T^{n+1}X$; (iii) follows by inductively applying Lemmas 2 and 4. Therefore, $R|_X \in \mathrm{bis}(\alpha)$.

Assume $R \in \mathrm{bis}(\alpha)$ and define $\underline{R} = \coprod_{i=0}^{n} R^{T^i}$. By construction of $\underline{R}$, $\underline{x}\ \underline{R}\ \underline{x'}$ implies that $\underline{x}, \underline{x'} \in T^i X$ for some $i \in \{0, \ldots, n\}$ meaning that the proof can be carried out by cases on each R^{T^i} composing $\underline{R}$. Assume $\underline{x}, \underline{x'} \in T^0 X = X$, then:

$$\begin{aligned} \underline{x}\ \underline{R}\ \underline{x'} &\iff \underline{x}\ R\ \underline{x'} \overset{(i)}{\implies} \alpha(\underline{x})\ R^{T^{n+1}}\ \alpha(\underline{x'}) \overset{(ii)}{\iff} \underline{\alpha}(\underline{x})\ (R^{T^n})^T\ \underline{\alpha}(\underline{x'}) \\ &\iff \underline{\alpha}(\underline{x})\ \underline{R}^T\ \underline{\alpha}(\underline{x'}) \end{aligned}$$

where (i) and (ii) follow by $R \in \mathrm{bis}(\alpha)$ and Lemma 2, respectively. Assume $\underline{x}, \underline{x'} \in T^{i+1}X$, we have that:

$$\begin{aligned} \underline{x}\ \underline{R}\ \underline{x'} &\iff \underline{x}\ R^{T^{i+1}}\ \underline{x'} \overset{(i)}{\iff} \underline{x}\ (R^{T^i})^T\ \underline{x'} \overset{(ii)}{\iff} \underline{\alpha}(\underline{x})\ (R^{T^i})^T\ \underline{\alpha}(\underline{x'}) \\ &\iff \underline{\alpha}(\underline{x})\ \underline{R}^T\ \underline{\alpha}(\underline{x'}) \end{aligned}$$

where (i) and (ii) follow by Lemma 2 and by definition of $\underline{\alpha}$ on $T^{i+1}X$. Therefore, $\underline{R} \in \mathrm{bis}(\overline{\alpha})$ and clearly $\underline{R}|_X = R$. □

Lemma 7 and its proof provide us with an encoding from systems whose steps are composed by multiple substeps to systems of substeps while preserving and reflecting their semantics in term of bisimulations. The trade-off of the encoding is a bigger statespace due to the explicit account of intermediate steps.

Lemma 8. *For $\mu\colon T \to S$ injective and $\alpha \in \mathsf{Sys}(T)$, $\mathrm{bis}(\alpha) = \mathrm{bis}(\mu_{\mathrm{car}(\alpha)} \circ \alpha)$.*

By applying the Lemma 8 to Example 1 we conclude that that bisimulations for ULTraSs coincide with bisimulations for WLTSs when these are seen as functional ULTraS as shown in [14, 15].

5 Reductions

In this section we formalize the intuition that a behaviour "shape" is (at least) as expressive as another whenever systems and homomorphisms of the latter can be "encoded" as systems and homomorphisms of the former, provided that their semantically relevant structures are preserved and reflected.

Definition 6. *For systems α and β, a* (system) reduction $\sigma\colon \alpha \to \beta$ *is given by a function $\sigma^c\colon \mathrm{car}(\alpha) \to \mathrm{car}(\beta)$ and a correspondence $\sigma^b \subseteq \mathrm{bis}(\alpha) \times \mathrm{bis}(\beta)$ s.t. σ^c carries a relation homomorphism for any pair of bisimulations in σ^b,* i.e.*:*

$$R\ \sigma^b\ R' \implies (x\ R\ x' \iff \sigma^c(x)\ R'\ \sigma^c(x')).$$

A system reduction $\sigma\colon \alpha \to \beta$ is called full *if $\sigma^c\colon \mathrm{car}(\alpha) \to \mathrm{car}(\beta)$ is surjective.*

For $\sigma\colon \alpha \to \beta$ a reduction, σ^c is always injective: the identity relation is always a bisimulation and hence condition (6) forces all x,x' such that $\sigma^c(x) = \sigma^c(x')$ to be equal in the beginning. Therefore the correspondence σ^b is always left-unique hence a surjection from $\mathrm{bis}(\beta)$ to $\mathrm{bis}(\alpha)$. This is indeed stronger than requiring preservation of bisimilarity since it entails that any bisimulation for α can be recovered by restricting some bisimulation for β to the image of $\mathrm{car}(\alpha)$ in $\mathrm{car}(\beta)$ through the map σ^c. Fullness implies σ^c and σ^b are isomorphism.

Remark 1. Condition (6) can be relaxed in two ways:
(a) $R\ \sigma^b\ R' \implies (x\ R\ x' \implies \sigma^c(x)\ R'\ \sigma^c(x'))$,
(b) $R\ \sigma^b\ R' \implies (x\ R\ x' \impliedby \sigma^c(x)\ R'\ \sigma^c(x'))$.
The condition (a) requires every bisimulation for α to be contained in some bisimulation for β whereas (b) requires every bisimulation for α to contain some bisimulation for β. Hence the two can be thought as *completeness* and *soundness* conditions for the reduction σ, respectively. □

System reductions can be extended to whole categories of systems provided they respect the structure of homomorphisms. Formally:

Definition 7. *For* C *and* D *categories of system, a* reduction σ *from* C *to* D*, written* $\sigma\colon \mathsf{C} \to \mathsf{D}$*, is a mapping that*

1. *assigns to any transition system* α *in* C *a system* $\sigma(\alpha)$ *in* D *and a system reduction* $\sigma_\alpha\colon \alpha \to \sigma(\alpha)$*;*
2. *assigns to any* $f\colon \alpha \to \beta$ *in* C *an homomorphism* $\sigma(f)\colon \sigma(\alpha) \to \sigma(\beta)$ *s.t.: (a)* $\sigma^c_\beta \circ f = \sigma(f) \circ \sigma^c_\alpha$*; (b)* $\sigma(id_\alpha) = id_{\sigma(\alpha)}$*; (c)* $\sigma(g \circ f) = \sigma(g) \circ \sigma(f)$*.*

A reduction $\sigma\colon \mathsf{C} \to \mathsf{D}$ *is called* full *if, and only if, every system reduction* σ_α *is full. A category* C *is said to* reduce *(resp.* fully *reduce) to* D*, if there is a reduction (resp. full reduction) from the* C *to* D*.*

Reductions can be easily composed at the level of their defining assignments. In particular, for reductions $\sigma\colon \mathsf{C} \to \mathsf{D}$ and $\tau\colon \mathsf{D} \to \mathsf{E}$, their composite reduction $\tau \circ \sigma\colon \mathsf{C} \to \mathsf{E}$ is a mapping that assigns to each system α the system $(\tau \circ \sigma)(\alpha)$ and the reduction given by $(\tau \circ \sigma)^c_\alpha \triangleq \tau^c_{\sigma(\alpha)} \circ \sigma^c_\alpha$ and $(\tau \circ \sigma)^b_\alpha \triangleq \tau^b_{\sigma(\alpha)} \circ \sigma^b_\alpha$; and to each $f\colon \alpha \to \alpha'$ the homorphism $(\tau \circ \sigma)(f)$. Reduction composition is associative and admits identities which are given on every C as the identity assignments for systems and homomorphisms. Any reduction restricts to a reduction from a subcategory of its domain and extends to a reduction to a super-category of its codomain. Moreover, fullness is preserved by the above operations.

For products, reductions can be given component-wise by suitable families of reductions that are "well-behaved" on homomorphisms. Formally:

Definition 8. *A family of reductions* $\{\sigma_i\colon \mathsf{C}_i \to \mathsf{D}_i\}_{i\in I}$ *is called* coherent *iff the following conditions hold for any* $i, j \in I$*:*

1. *if a function* f *extends to* $f_i \in \mathsf{C}_i$ *then there is* $f_j \in \mathsf{C}_j$ *s.t.* f *extends to* f_j*;*
2. $\sigma_i(f_i)$ *and* $\sigma_j(f_j)$ *share their underlying function whenever* f_i *and* f_j *do.*

Theorem 2. *A coherent family of (full) reductions* $\{\sigma_i\colon \mathsf{Sys}(T_i) \to \mathsf{Sys}(S_i)\}_{i\in I}$ *defines a (full) reduction* $\sigma\colon \mathsf{Sys}(\prod_{i\in I} T_i) \to \mathsf{Sys}(\prod_{i\in I} S_i)$*.*

Proof. Assume $\{\sigma_i\}_{i\in I}$ as above. For $\alpha \in \mathsf{Sys}(\prod T_i)$ let $\alpha_i = \pi_i \circ \alpha$ and define

$$\sigma(\alpha) \triangleq \langle \dots, \sigma_i(\alpha_i), \dots \rangle \qquad \sigma^c_\alpha \triangleq \sigma^c_{i,\alpha_i} \qquad \sigma^b_\alpha \triangleq \bigcap_{i\in I} \sigma^b_{i,\alpha_i}$$

The assignment extends to all systems in $\mathsf{Sys}(\prod_{i\in I} T_i)$ and is well-defined by coherency and Lemma 6 since $R\ \sigma^b_\alpha\ R' \iff \forall i \in I(R\ \sigma^b_{i,\alpha_i}\ R')$ and for all $i \in I$, $\sigma^c_\alpha = \sigma^c_{i,\alpha_i}$. For any $i \in I$, $f\colon \alpha \to \beta$ defines an homomorphism $f_i\colon \alpha_i \to \beta_i$ in $\mathsf{Sys}(T_i)$ sharing its underlying function. Define $\sigma(f)$ as the homomorphism arising from the function underlying $\sigma_i(f_i)$. By coherency, the mapping is well-defined and satisfies all the necessary conditions since all σ_i are reductions. □

Correspondences for bisimulations presented in Lemmas 7 and 8 extend to reductions: injective transformations define full reductions and homogeneous systems reduce to systems for the base endofunctor, as formalised below.

Theorem 3. *For $\mu\colon T \to S$ an injective transformation, there is a full reduction $\hat{\mu}\colon \mathsf{Sys}(T) \to \mathsf{Sys}(S)$ given, on each α and each $f\colon \alpha \to \beta$ as $\hat{\mu}(\alpha) \triangleq \mu_{\mathrm{car}(\alpha)} \circ \alpha$, $\hat{\mu}^c_\alpha \triangleq id_{\mathrm{car}(\alpha)}$, $\hat{\mu}^b_\alpha \triangleq id_{\mathrm{bis}(\alpha)}$, and $\hat{\mu}(f) \triangleq f$.*

This theorem allows us to formalise the hierarchy shown in Section 1. For instance, the transformation described in Example 1 defines a full reduction from WLTSs to ULTraSs. Probabilistic systems are covered by the transformation induced by the inclusion $\mathcal{D}X \subseteq \mathcal{F}_{[0,\infty)}X$ whereas the remaining cases are trivial.

Theorem 4. *If T preserves injections then $\mathsf{Sys}(T^{n+1})$ reduces to $\mathsf{Sys}(T)$.*

Proof. Recall from Lemma 7 the construction of $\underline{\alpha}\colon \underline{X} \to T\underline{X}$ for any $\alpha\colon X \to T^{n+1}X$ and let $\iota_0\colon X \to \underline{X}$ denote the obvious injection. Define $\sigma\colon \mathsf{Sys}(T^{n+1}) \to \mathsf{Sys}(T)$ as the reduction given on each transition system α in $\mathsf{Sys}(T^{n+1})$ as

$$\sigma(\alpha) \triangleq \underline{\alpha} \quad \sigma^c_\alpha \triangleq \iota_0 \quad \sigma^b_\alpha \triangleq \{(R, \underline{R}) \mid R = \underline{R}|_X,\ R \in \mathrm{bis}(\alpha),\ \underline{R} \in \mathrm{bis}(\underline{\alpha})\}$$

and on each homomorphism $f\colon \alpha \to \beta$ in $\mathsf{Sys}(T^{n+1})$ as $\sigma(f) \triangleq \coprod_{i=0}^{n} T^i f$. By Lemma 7, σ^b_α is a correspondence and by construction σ respects homomorphism composition and identities. Thus, σ is a reduction from $\mathsf{Sys}(T^{n+1})$ to $\mathsf{Sys}(T)$. □

6 Application: reducing FuTSs to WLTSs

In this section we apply the theory presented in the previous sections to prove that (categories of) FuTSs reduce to (categories of) simple FuTSs, *i.e.* WLTSs. The reduction is given in stages reflecting the endofunctors structure.

Definition 9. *A monoid sequence $\vec{M}$ is called* homogeneous *if its elements are the same. FuTSs on $\vec{M}$ are called* homogeneous *if $\vec{M}$ is homogeneous.*

Lemma 9. *The FuTSs category fully reduces to that of homogeneous FuTSs.*

Proof. For a sequence of monoids $\vec{M} = \langle M_0, \dots, M_n \rangle$ let N denote the product monoid $\prod_{i=0}^{n} M_i$. Let 0_j denote the unit of M_j. For each $i \in \{0, \dots, n\}$, the assignment $x \mapsto \langle 0_0, \dots, 0_{i-1}, x, 0_{i+1}, \dots 0_n \rangle$ extends to an injective monoid homomorphism $m_i \colon M_i \to N$. The assignment $\phi \mapsto m_i \circ \phi$ defines an injective natural transformation $\mathcal{F}_{M_i} \to \mathcal{F}_N$ which extends to an injective transformation $\mathcal{F}_{\vec{M}} \to \mathcal{F}_{\vec{N}}$. We conclude by Theorems 2 and 3. □

Lemma 10. *The nested FuTSs category reduces to that of simple FuTSs.*

Proof. By Lemma 9 and Theorems 2 and 4. □

Lemma 11. *The FuTSs category reduces to that of combined FuTSs.*

Proof. By Theorem 2 and Lemma 10. □

Lemma 12. *The combined FuTSs category fully reduces to that of simple FuTSs.*

Proof. For a sequences $\vec{A} = \langle A_0, \dots, A_n \rangle$ and $\vec{M} = \langle M_0, \dots, M_n \rangle$ let B and N denote the cartesian product $\prod_{i=0}^{n} A_i$ and the product monoid $\prod_{i=0}^{n} M_i$, respectively. The mapping $\langle \phi_0, \dots, \phi_n \rangle \mapsto \lambda \langle a_0, \dots, a_n \rangle . \lambda x . \langle \phi_0(a_0)(x), \dots, \phi_n(a_n)(x) \rangle$ extends to an injective natural transformation from $(\mathcal{F}_{\vec{M}} -)^{\vec{A}} = \prod_{i=0}^{n} (\mathcal{F}_{M_i} -)^{A_i}$ to $(\mathcal{F}_N -)^B$. We conclude by Theorem 3. □

Theorem 5. *The FuTSs category reduces to that of simple FuTSs* i.e. *WLTSs.*

Proof. By Lemmas 11 and 12. □

For instance, consider an ULTraS $\alpha \colon X \to (\mathcal{P}_f \mathcal{F}_M X)^A$. By Lemma 9 it fully reduces to a homogeneous nested FuTS (X, α') for the sequences of labels and monoids $\langle A \rangle$ and $\langle \langle \mathbb{B} \times M, \mathbb{B} \times M \rangle \rangle$, respectively and such that:

$$\alpha'(x)(a)(\phi) \triangleq \begin{cases} \langle \mathtt{tt}, 0 \rangle & \text{given } \psi \in \alpha(x)(a) \text{ s.t. } \phi(y) = \langle \psi(y), 0 \rangle \text{ for all } y \in X \\ \langle \mathtt{ff}, 0 \rangle & \text{otherwise} \end{cases}$$

By Lemma 10, α' reduces to the WLTS $(X + \mathcal{F}_{\mathbb{B} \times M} X, \underline{\alpha'})$ with labels from A, weights from $\mathbb{B} \times M$, and such that:

$$\underline{\alpha'}(y)(a)(y') \triangleq \begin{cases} y(y') & \text{if } y \in \mathcal{F}_{\mathbb{B} \times M} X \text{ and } y' \in X \\ \alpha'(y)(a)(y') & \text{if } y \in X \text{ and } y' \in \mathcal{F}_{\mathbb{B} \times M} X \\ \langle \mathtt{ff}, 0 \rangle & \text{otherwise} \end{cases}$$

As exemplified by the above reduction for ULTraSs, FuTSs can be reduced to WLTSs by extending the original state space with weight functions and splitting steps accordingly. From this perspective, weight functions are *hidden states* in the original systems which the proposed reduction renders explicit. This observation highlights a trade-off between state and behaviour complexity of these semantically equivalent meta-models.

7 Conclusions

In this paper we have introduced a notion of *reduction* for categories of discrete state transition systems, and some general results for deriving reductions from the shape of computational aspects. As an application of this theory we have shown that FuTSs reduce to WLTSs, thus collapsing the upper part of the hierarchy in Section 1. Besides the classification interest, this result offers a solid bridge for porting existing and new results from WLTSs to FuTSs. For instance, SOS specifications formats presented in [10, 15] can cope now with FuTSs, and any abstract GSOS for these systems admits a specification in the format presented in [15]. Likewise, developing an HML style logic for bisimulation on WLTSs would readily yield a logic capturing bisimulation on FuTSs.

It remains an open question whether the hierarchy can be further collapsed, especially when other notion of reduction are considered. In fact, requiring a correspondence between bisimulations for the original and reduced systems may be too restrictive in some applications like bisimilarity-based verification techniques. This suggests to investigate laxer notions of reductions, such as those indicated in Remark 1. Another direction is to consider different behavioural equivalences, like trace equivalence or weak bisimulation. We remark that, as shown in [3, 4, 7], in order to deal with these and similar equivalences, endofunctors need to be endowed with a monad (sub)structure; although WLTSs are covered in [3, 13], an analogous account of FuTSs is still an open problem.

References

1. P. Aczel and N. Mendler. A final coalgebra theorem. In *Proc. CTCS*, volume 389 of *LNCS*, pages 357–365. Springer, 1989.
2. M. Bernardo, R. De Nicola, and M. Loreti. A uniform framework for modeling nondeterministic, probabilistic, stochastic, or mixed processes and their behavioral equivalences. *Inf. Comput.*, 225:29–82, 2013.
3. T. Brengos, M. Miculan, and M. Peressotti. Behavioural equivalences for coalgebras with unobservable moves. *JLAMP*, 84(6):826–852, 2015.
4. T. Brengos and M. Peressotti. A Uniform Framework for Timed Automata. In *Proc. CONCUR*, volume 59 of *LIPIcs*, pages 26:1–26:15, 2016.
5. R. De Nicola, D. Latella, M. Loreti, and M. Massink. A uniform definition of stochastic process calculi. *ACM Computing Surveys*, 46(1):5, 2013.
6. R. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121:130–141, 1990.
7. I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *LMCS*, 3(4), 2007.
8. H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.
9. J. Hillston. *A compositional approach to performance modelling*. Cambridge, 1996.
10. B. Klin and V. Sassone. Structural operational semantics for stochastic and weighted transition systems. *Inf. Comput.*, 227:58–83, 2013.
11. D. Latella, M. Massink, and E. de Vink. Bisimulation of labelled state-to-function transition systems coalgebraically. *LMCS*, 11(4), 2015.

12. D. Latella, M. Massink, and E. de Vink. A definition scheme for quantitative bisimulation. In *QAPL*, volume 194 of *EPTCS*, pages 63–78, 2015.
13. M. Miculan and M. Peressotti. Weak bisimulations for labelled transition systems weighted over semirings. *CoRR*, abs/1310.4106, 2013.
14. M. Miculan and M. Peressotti. GSOS for non-deterministic processes with quantitative aspects. In *Proc. QAPL*, volume 154 of *EPTCS*, pages 17–33, 2014.
15. M. Miculan and M. Peressotti. Structural operational semantics for non-deterministic processes with quantitative aspects. To appear in *TCS*, 2016.
16. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
17. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249(1):3–80, 2000.
18. R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.
19. S. Staton. Relating coalgebraic notions of bisimulation. *LMCS*, 7(1), 2011.

A Omitted proofs

Proof of Lemma 1 For f injective, the assignments

$$\psi \mapsto f \circ \psi \qquad Z \mapsto \{f(z) \mid z \in Z\} \qquad \phi \mapsto \lambda y. \sum_{x:f(x)=y} \phi(x)$$

describe injective functions. □

Proof of Lemma 2 Let $\kappa^S \colon SX \to SX/R^S$ be the canonical projection to the quotient induced by the equivalence relation R^S. Since, by definition, $\kappa^S(\rho) = \kappa^S(\theta)$ implies $(S\kappa)(\rho) = (S\kappa)(\theta)$ there is a (unique) function $q^S \colon SX/R^S \to S(X/R)$ such that $S\kappa = q^S \circ \kappa^S$. From $TS\kappa = Tq^S \circ T\kappa^S$ and the definition of R^{TS} and $\left(R^S\right)^T$, it follows that: $\phi \left(R^S\right)^T \psi \implies (TS\kappa)(\phi) = (TS\kappa)(\psi) \implies \phi\ R^{TS}\ \psi$ proving first part of the thesis. Since $\rho\ R^S\ \theta \iff \kappa^S(\rho) = \kappa^S(\theta)$ we conclude that q^S is an injection and, by hypothesis, Tq^S is an injection too. Therefore:

$$\begin{aligned} \phi\ R^{TS}\ \psi &\implies (TS\kappa)(\phi) = (TS\kappa)(\psi) \implies (T\kappa^S)(\phi) = (T\kappa^S)(\psi) \\ &\implies \phi \left(R^S\right)^T \psi \end{aligned}$$

completing the proof. □

Proof of Lemma 3 Write T for $\prod_{i\in I} T_i$ and recall that $\left(\prod_{i\in I} T_i\right) X$ is $\prod_{i\in I} T_i X$. Then:

$$\begin{aligned} \phi\ R^T\ \psi &\iff (\textstyle\prod T_i\kappa)(\phi) = (\prod T_i\kappa)(\psi) \iff \prod (T_i\kappa)(\phi_i) = \prod (T_i\kappa)(\psi_i) \\ &\iff \phi \textstyle\prod R^{T_i}\ \psi \end{aligned}$$

where $\kappa \colon X \to X/R$ is the canonical projection to the quotient induced by R and $\pi_i \colon \prod_{i\in I} T_i X \to T_i X$ is the i-th projection. □

Proof of Lemma 4 Let $\kappa \colon Y \to Y/R$ and $\kappa' \colon X \to X/R|_X$ be the canonical projections induced by R and $R|_X$, respectively. Since the latter is given by restriction of the former to $X \subseteq Y$, there is a unique and injective map $q \colon X/R|_X \to Y/R$ such that $\kappa = q \circ \kappa$. The first part of the thesis follows by:

$$\begin{aligned} \phi \left(R|_X\right)^T \psi &\implies (T\kappa')(\phi) = (T\kappa')(\psi) \implies (T\kappa)(\phi) = (T\kappa)(\psi) \\ &\implies \phi\ R^T\ \psi \quad \text{since } T\kappa = Tq \circ T\kappa'. \end{aligned}$$

On the other hand, by hypothesis on T, Tq is injective and hence

$$\begin{aligned} \phi\ R^T|_{TX}\ \psi &\implies (T\kappa)(\phi) = (T\kappa)(\psi) \implies (T\kappa')(\phi) = (T\kappa')(\psi) \\ &\implies \phi \left(R|_X\right)^T \psi \end{aligned}$$

□

Proof of Lemma 5 It holds that

$$\begin{aligned}\phi\ R^T\ \psi &\iff T\kappa(\phi) = T\kappa(\psi) \overset{(i)}{\iff} (\mu_X \circ T\kappa)(\phi) = (\mu_X \circ T\kappa)(\psi)\\ &\overset{(ii)}{\iff} (S\kappa \circ \mu_X)(\phi) = (S\kappa \circ \mu_X)(\psi) \iff \mu_X(\phi)\ R^S\ \mu_X(\psi)\end{aligned}$$

where (i) and (ii) follow by μ_X being injective and by μ being a natural transformation, respectively. □

Proof of Theorem 1 Assume R is a bisimulation for $\alpha\colon X \to TX$. For $f, f'\colon X \to Y$ s.t. $x\ R\ x' \implies f(x) = f'(x')$ we have that:

$$\begin{aligned}x\ R\ x &\implies \alpha(x)\ R^T\ \alpha(x') \iff (T\kappa \circ \alpha)(x) = (T\kappa \circ \alpha)(x')\\ &\overset{(i)}{\implies} (Tf \circ \alpha)(x) = (Tf' \circ \alpha)(x')\end{aligned}$$

where (i) follows by noting that, since $\kappa\colon X \to X/R$ is a canonical projection and $x\ R\ x' \implies f(x) = f'(x')$, there is (a unique) $q\colon X/R \to Y$ s.t. $f = q \circ \kappa = f'$.

Assume R is a precongruence for α, we have that:

$$\begin{aligned}x\ R\ x &\overset{(i)}{\iff} \kappa(x) = \kappa(x') \overset{(ii)}{\implies} (T\kappa \circ \alpha)(x) = (T\kappa \circ \alpha)(x')\\ &\overset{(iii)}{\iff} \alpha(x)\ R^T\ \alpha(x')\end{aligned}$$

where (i) follows by definition of $\kappa\colon X \to X/R$, (ii) by R being a precongruence, and (iii) by definition of R^T. □

Proof of Corollary 1 By Theorem 1 and [19, Thm. 4.1]. □

Proof of Lemma 6 By Theorem 2, $\alpha(x)\ R^T\ \alpha(x') \iff \alpha(x)\ \left(\prod_{i\in I} R^{T_i}\right)\ \alpha(x')$ and hence $\alpha(x)\ R^T\ \alpha(x') \iff \forall i \in I\ (\pi_i\alpha)(x)\ R_i^T\ (\pi_i\alpha)(x')$. □

Proof of Lemma 8 For a relation R, $R \in \mathrm{bis}(\alpha)$ iff $x\ R\ y \implies \alpha(x)\ R^T\ \alpha(y)$ and $R \in \mathrm{bis}(\mu_X \circ \alpha)$ iff $x\ R\ y \implies (\mu_X \circ \alpha)(x)\ R^S\ (\mu_X \circ \alpha)(y)$. We conclude by Lemma 5. □

Proof of Theorem 3 By Lemma 8. □

A Locally Connected Spanning Tree Can Be Found in Polynomial Time on Simple Clique 3-Trees*

Tiziana Calamoneri, Matteo Dell'Orefice, and Angelo Monti

Computer Science Department,
"Sapienza" University of Rome, Italy
calamo/monti@di.uniroma1.it, matteodellorefice@gmail.com

Abstract. A locally connected spanning tree (LCST) T of a graph G is a spanning tree of G such that for each node its neighborhood in T induces a connected subgraph in G. The problem of determining whether a graph contains an LCST or not has been proved to be NP-complete, even if the graph is planar or chordal. The main result of this paper is a linear time algorithm that, given an SC 3-tree (i.e. a maximal planar chordal graph), determines in linear time whether it contains an LCST or not, and produces one if it exists. We give an analogous result even for the case when the input graph is an SC 2-tree (i.e. a maximal outerplanar graph).

Keywords: locally connected spanning tree, SC k-trees, 2-trees, chordal graphs, planar graphs.

1 Introduction

A *locally connected spanning tree (LCST)* T of a graph G is a spanning tree of G such that for each node its neighborhood in T induces a connected subgraph in G [3]. It is well known that an interconnection network can be modeled as a graph and, in this context, the existence of such a spanning tree ensures, in case of site failures, effective communication among operative sites as long as these failures are isolated [14].

Cai proved in [4] that the problem of determining whether a graph contains an LCST is NP-complete even when the input graph is restricted to be planar or split (and, *a fortiori*, chordal). So, researchers have looked for special classes of graphs for which the problem is polynomially solvable.

* Partially supported by the Italian Ministry of Education and University, PRIN project "AMANDA: Algorithmics for MAssive and Networked DAta"

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 103–121 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

In particular, in [4] the problem has been proven to admit a linear solution on directed path graphs, a superclass of interval graphs; this result has been first generalized to the superclass of strongly chordal graphs [7] and then further extended to doubly chordal graphs [10]. Moreover, in [8] the authors present a linear time algorithm to solve the problem on circular arc graphs, a natural superclass of interval graphs. Finally, linear time algorithms for the LCST problem on cographs and co-bipartite graphs are provided in [10]. For a visual summary of the known results, see Figure 1.

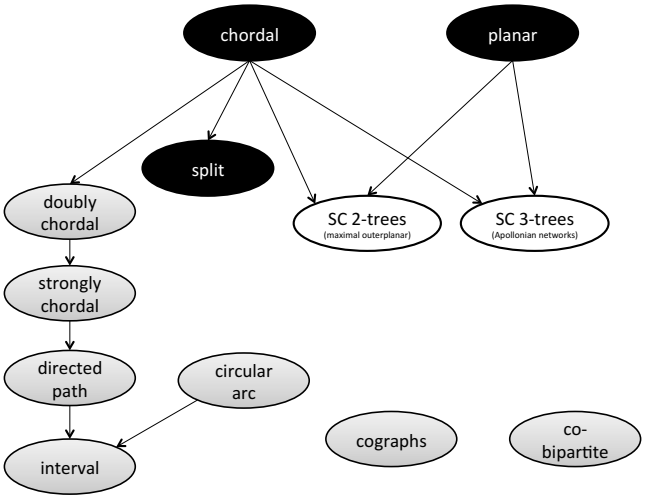


Fig. 1. The state of the art concerning the complexity of the LCST problem. The problem is NP-complete on black classes and linearly solvable on grey classes. White classes are studied in this paper, and for them a linear time algorithm is designed.

In this paper we consider the SC 3-trees (i.e. maximal planar chordal graphs) - an interesting and naturally defined subclass of k-trees introduced in [9] - and we prove that the problem of finding an LCST is linearly solvable when restricted to them. We give an analogous result even for the case when the input graph is an SC 2-tree (i.e. a maximal outerplanar graph).

The rest of this paper is organized as follows: Section 2 is devoted to recall some known notions and to state some preliminary results that will be useful in the successive two sections. Section 3 is devoted to give a linear time algorithm for finding an LCST on an SC 2-tree; it is preliminary to Section 4, where the idea provided in the previous section is generalized and refined in order to prove that an LCST of an SC 3-tree can be found in linear time, if it exists. Finally, Section 5 concludes the paper addressing some open problems.

2 Preliminaries

In this section we recall some known notions, and state some preliminary lemmas for SC k-trees that will be useful in the main part of this paper restricted to $k = 2, 3$.

Definition 1. [9] *Given a positive integer k,* simple-clique k-trees *(in short* SC k-trees*) are recursively defined as follows:*

- *The complete graph with $k+1$ nodes is an SC k-tree.*
- *An SC k-tree with $n+1$ nodes ($n \geq k+1$) can be constructed from an SC k-tree with n nodes by adding a node adjacent to all nodes of a k-clique not previously chosen in the existing SC k-tree, and only to these nodes.* □

In this paper we deal with the two classes of SC 2-trees and SC 3-trees. SC 2-trees coincide with maximal outerplanar graphs [9] while SC 3-trees coincide with Apollonian networks [1], and are in fact equivalent to the intersection class of chordal and maximal planar graphs [9]. Hence, these two classes represent interesting subclasses of both chordal and planar graphs.

Definition 2. [5] *Given a graph $G = (V, E)$ and two non-adjacent nodes u and v of V, a subset $S \subseteq V \setminus \{u, v\}$ is an* (u, v)-separator *if the removal of S from G separates u and v into distinct connected components.*

Let S be an (u, v)-separator of G. S is a *minimal (u, v)-separator* if no proper subset of S separates u from v. More generally, S is a *minimal separator* if it is a minimal (u, v)-separator, for some pair (u, v) of non adjacent nodes.

Given a set of nodes $V' \subseteq V$ of a graph G, we denote by $G[V']$ the *subgraph induced* in G by the nodes in V'.

We now give some properties of a minimal separator of an SC k-tree. Due to lack of space, the proofs are deferred to the Appendix.

Lemma 1. *Let G be an SC k-tree, and S be a minimal separator in G, then the graph $G[V \setminus S]$ has exactly two connected components A_S and B_S and the two graphs $G[A_S \cup S]$ and $G[B_S \cup S]$ are SC k-trees.*

Proof. see Appendix. □

From now on, fixed a minimal separator S, we will continue to call A_S and B_S the two connected components of $G \setminus S$.

In the following lemma we recall some simple properties of SC k-trees that can be proved by induction on the number of nodes in G and that have been stated either in [12] or in [11] for the more general class of k-trees.

Lemma 2. *Let G be an SC k-tree, then*

(i) G has $(k+1)$-cliques but no $(k+2)$-cliques,
(ii) every minimal separator of G is a k-clique,

(iii) *G is a chordal graph,*
(iv) *For each k-clique K in G there exists a node t such that $K \cup \{t\}$ induces a $(k+1)$-clique in G.*

Lemma 3. *Let G be an SC k-tree, then for any minimal separator S in G there are two nodes a and b, such that S is an (a,b)-separator; moreover $S \cup \{a\}$ and $S \cup \{b\}$ are $(k+1)$-cliques in G.*

Proof. see Appendix. □

We now recall the following generalization of line graphs introduced in [6].

Definition 3. *The* k-line graph *of a graph G, in short $\mathcal{L}_k(G)$, is defined as a graph whose nodes are the k-cliques in G. Two distinct such nodes are adjacent in the k-line graph if and only if they have $k-1$ nodes in common in G.*

In the following, for a node X in $\mathcal{L}_k(G)$, with a small abuse of notation, we will denote by X also the set of nodes of G that are in the the k-clique corresponding to X.

Lemma 4. *Let G be an SC k-tree, then a k-clique S in G is a minimal separator if and only if there exist two adjacent nodes X_1 and X_2 in $\mathcal{L}_{k+1}(G)$ such that $S = X_1 \cap X_2$.*

Proof. see Appendix. □

The k-line graphs have been used to obtain the following characterization of SC k-trees:

Theorem 1. [9] *A k-tree G is an SC k-tree if and only if the $(k+1)$-line graph $\mathcal{L}_{k+1}(G)$ of G is a tree.*

An SC k-tree whose $(k+1)$-line graph $\mathcal{L}_{k+1}(G)$ is a path is called *k-path*.

Since our algorithms exploit the $(k+1)$-line graph, we are interested to output $\mathcal{L}_{k+1}(G)$ in linear time from an SC k-tree input graph G; this is possible, as shown by the following result.

Lemma 5. *Let G be an SC k-tree; then tree $\mathcal{L}_{k+1}(G)$ can be computed in linear time.*

Proof. see Appendix.

Given a graph G, and one of its spanning trees T, for each node v of G, $N_T(v)$ represents the set of the nodes of G that are adjacent to v in T; these nodes will be called *T-neighbors of v*. The next lemma states a necessary condition that an LCST of a SC k-tree satisfies. As we will see later, 3 states the sufficiency of this condition for the case $k = 3$.

Lemma 6. *Let G be an SC k-tree, $k \geq 2$, S be one of its minimal separators and T be an LCST in G. We have that:*

(i) *if $T[S]$ contains an isolated node, then its T-neighbors completely lie either in A_S or in B_S.*
(ii) *$G[S]$ contains at least one edge of T.*

Proof. see Appendix.

3 An Algorithm to Determine an LCST of an SC 2-Tree

Cai [3] proved that a nontrivial graph contains an LCST if and only if it contains a spanning 2-tree T such that T does not contain as induced subgraph a 3-*sun*. The 3-sun graph is made of a central triangle, and three independent nodes, each adjacent to both ends of a single edge of the triangle.

Corollary 1. *An SC 2-tree G contains an LCST if and only if G does not contain as induced subgraph a* 3-sun.

From the characterization above, we deduce the structure of the subclass of SC 2-trees admitting an LCST (i.e. 2-paths, in Lemma 7 and Corollary 2), and from there we look for the actual edges of the existing LCST (Lemma 8).

Lemma 7. *Let G be an SC 2-tree. Its 3-line graph $\mathcal{L}_3(G)$ has nodes of degree 3 if and only if G contains a 3-sun as induced subgraph.*

Proof. see Appendix.

Now, since an SC k-tree G is a k-path if and only if $\mathcal{L}_{k+1}(G)$ is a path, from the above two results we have:

Corollary 2. *An SC 2-tree G contains an LCST if and only if $\mathcal{L}_3(G)$ is a path.*

Now we give a characterization of LCSTs of 2-paths. This characterization allows us to design an algorithm that finds an LCST of a 2-path in linear time.

Lemma 8. *Let G be a 2-path, and T be one of its spanning trees. T is an LCST if and only if, for each minimal separator $S = \{x, y\}$ of G the edge xy is in T.*

Proof. see Appendix.

From the above results it is easy to obtain a linear time algorithm that, given an n node SC 2-tree G, return an LCST of G if it exists, returns 'no' otherwise.

Algorithm FindLCSTinSC2trees
Input: an n node SC 2-tree G;
Output: an LCST of G if it exists, NO otherwise.
Compute tree $\mathcal{L}_3(G)$;
if $\mathcal{L}_3(G)$ is not a path **then** return *no*;
Let $X_1, X_2, \ldots X_{n-2}$ be a linear order of the nodes of the path $\mathcal{L}_3(G)$;
$T \leftarrow \emptyset$;
if $\mathcal{L}_3(G)$ consists of a single node X_1 **then**
    insert in T any two edges of the 3-clique X_1 and return T;
for $i = 1$ **to** $n - 3$ **do**
    add to T the edge in the minimal separator $X_i \cap X_{i+1}$;
Let $X_j = \{x_j, y_j, z_j\}$ for $j \in \{1, n-2\}$;
Let x_1y_1 be the edge in T for $X_1 \cap X_2$ and $x_{n-3}y_{n-3}$ be the edge in T for $X_{n-3} \cap X_{n-2}$;
Add to T the two edges z_1x_1 and $z_{n-2}x_{n-2}$;
return T.

Theorem 2. *(Correctness and Complexity) Algorithm* **FindLCSTinSC2trees** *determines an LCST of a given SC 2-tree if and only if it exists and runs in linear time.*

Proof. If $\mathcal{L}_3(G)$ is not a path the algorithm, in agreement with Lemma 2, correctly returns "no".

It remains to show that the tree T constructed by the algorithm visiting the path $\mathcal{L}_3(G)$ is an LCST of G. This easily follows noting that the algorithm constructs the LCST T exploiting the characterization in Lemma 8 and selects all edges induced by each minimal separator of G. Note that, after the selection in T of the $n-2$ minimal separators of G, it remains to connect to T the only two nodes of degree 2, z_1 and z_{n-2}, that occur in G. The node z_1 is connected in T to a node of the minimal separator $X_1 \cap X_2$ while the node z_{n-2} is connected in T to a node of of the minimal separator $X_{n-3} \cap X_{n-2}$. It is easy to see that the resulting spanning tree of G is an LCST.

For what concerns the time complexity, observe that $\mathcal{L}_3(G)$ can be computed in linear time (cf. Lemma 5) and the same asymptotic time is sufficient also to traverse the $n-2$ nodes of the path $\mathcal{L}_3(G)$ to gather the edges of T. □

4 An Algorithm to Determine an LCST of an SC 3-Tree

In the previous section, we have seen that it is easy to determine an LCST of an SC 2-tree G, if it exists, exploiting its $\mathcal{L}_3(G)$. Unfortunately, when we move to SC 3-trees, things seem to be not so easy anymore. Nevertheless, we will show that it is possible to determine an LCST of an SC 3-tree G, if it exists, exploiting its $\mathcal{L}_4(G)$, in linear time. This is the aim of this section. Lemmas 9 and 10 are technical statements needed for the proof of Theorem 3, which gives a necessary and sufficient condition for the existence of an LCST in an SC 3-tree.

In the following statement, the graph $2K_2$ is the disjoint union of two copies of K_2.

Lemma 9. *Let G be an SC 3-tree and T be one of its spanning trees. If, for each minimal separator $S = \{x, y, z\}$ of G, one of the following is true:*

(i) *T contains exactly two edges of $G[S]$*
(ii) *T contains exactly one edge of $G[S]$ (w.l.o.g. this edge is xy) and either $N_T(z) \subseteq A_S$ or $N_T(z) \subseteq B_S$*

then, for each node X in $\mathcal{L}_4(G)$ it holds $T[X] \neq 2K_2$.

Proof. see Appendix.

Lemma 10. *Let G be an SC 3-tree and T be one of its spanning trees. If, for each minimal separator $S = \{x, y, z\}$ of G, the following is true:*

(ii) *T contains exactly one edge of $G[S]$ (w.l.o.g. this edge is xy) and either $N_T(z) \subseteq A_S$ or $N_T(z) \subseteq B_S$*

then G is a 3-path and T is an LCST.

Proof. see Appendix.

Theorem 3. *Let G be an SC 3-tree, and T be one of its spanning trees. T is an LCST if and only if, for each minimal separator $S = \{x, y, z\}$, one of the following is true:*

(i) T contains exactly two edges of $G[S]$;
(ii) T contains exactly one edge of $G[S]$ (w.l.o.g. this edge is xy) and either $N_T(z) \subseteq A_S$ or $N_T(z) \subseteq B_S$.

Proof. We prove the two implications separately.

($\Rightarrow$) Let T be an LCST, and let us prove that either *(i)* or *(ii)* hold on S.

First, notice that from item *(ii)* of Lemma 2, $G[S]$ is a 3-clique, so it cannot contain three edges of T, otherwise a cycle would occur in T; so, in view of Lemma 6, $G[S]$ contains either one or two edges of T. If it contains exactly two edges of T, then *(i)* holds and we have done. If, on the contrary, $G[S]$ contains exactly one edge xy of T then, by Lemma 6, $N_T(z)$ has an empty intersection either with A_S or with B_S, that is *(ii)* holds.

($\Leftarrow$) Let us now assume that S satisfies either *(i)* or *(ii)*, and let us prove that T is an LCST. The proof proceeds by induction on the number n of nodes of G. For $n = 4$ (the basis of the induction) G is a 4-clique and each spanning tree of G is an LCST and the claim is trivially true since no separator exists. Assume now that G has $n > 4$ nodes and the claim is true for every SC 3-tree with less than n nodes. If all the minimal separators of G satisfy (ii), by Lemma 10 we have that G is a 3-path and T is an LCST.

It remains to consider the case in which there exists a separator $\tilde{S}$ of G that satisfies *(i)*. Consider graphs $G_1 = G[A_{\tilde{S}} \cup \tilde{S}]$ and $G_2 = G[B_{\tilde{S}} \cup \tilde{S}]$ and the spanning trees $T_1 = T[A_{\tilde{S}} \cup \tilde{S}]$ of G_1 and $T_2 = T[B_{\tilde{S}} \cup \tilde{S}]$ of G_2. In view of Lemma 1, graphs G_1 and G_2 are SC 3-trees and each separator of one of these two graphs is in fact a separator of G hence, for each separator S of G_i, $1 \leq i \leq 2$, the tree T_i satisfies either *(i)* or *(ii)*. By inductive hypothesis, it follows that T_1 and T_2 are LCSTs of G_1 and G_2, respectively. Moreover, we have

$$N_T(u) = \begin{cases} N_{T_1}(u) & \text{if } u \in A_{\tilde{S}} \\ N_{T_2}(u) & \text{if } u \in B_{\tilde{S}} \\ N_{T_1}(u) \cup N_{T_2}(u) & \text{if } u \in \tilde{S} \end{cases}$$

For each u not in $\tilde{S}$, we already know that its T-neighbors are connected in $G \setminus \{u\}$ (since T_1 is an LCST of G_1 and T_2 in an LCSTs of G_2); for each u in $\tilde{S}$, its T-neighbors are partially in $A_{\tilde{S}}$ (and they are connected in $A_{\tilde{S}} \cup \tilde{S}$), partially in $A_{\tilde{S}}$ (and they are connected in $B_{\tilde{S}} \cup \tilde{S}$), and partially in $\tilde{S}$ (through which all the T-neighbors of u are connected since $\tilde{S}$ is a 3-clique in G). It follows that T is an LCST of G. □

The next three definitions aim to introduce the concept of *partial solution* and their *labels*, which will be the crucial operating principle of the algorithm. A

partial solution is simply a "piece" of LCST of a peripheral portion of the graph in input, and those solutions are combined together at each iteration, if possible.

Definition 4. *Let $S = \{x, y, z\}$ be a minimal separator of an SC 3-tree G. Let G' be the graph obtained from G by substituting set B_S with the single node $\{b\}$ connected to all the three nodes in S. A* partial solution on $A_S \cup S$ w.r.t. S, *$H_{A_S \cup S}$, is a spanning forest of $A_S \cup S$ such that there exists an LCST T of G' such that $H_{A_s \cup S} = T[A_S \cup S]$.*

In the following we will call simply H a partial solution when S and $A_S \cup S$ are clear from the context.

Theorem 3, characterizing LCSTs in SC 3-trees, suggests that partial solutions fall in exactly three distinct categories, depending on how many edges of H are induced in S, and depending on the presence or not of an isolated node in H. The next definition formalizes this fact.

Definition 5. *Let $S = \{x, y, z\}$ be a minimal separator of an SC 3-tree G. Let H be a partial solution on $A_S \cup S$ w.r.t. S. We say that H has label:*

- *α_x if yz is an edge of H and x is isolated in H;*
- *β_x if xy and xz are both in H;*
- *γ_x if yz is an edge of H, xy and xz are not in H, and x is not isolated in H.*

Analogous definitions can be given for labels α_y, β_y and γ_y, and α_z, β_z and γ_z.

Definition 6. *Let S be a minimal separator in G, and assume $|A_S| = 1$. The canonical partial solutions of $G[A_S \cup S]$ associated to label χ_v (with $\chi \in \{\alpha, \beta, \gamma\}$ and $v \in S$) are depicted below.*

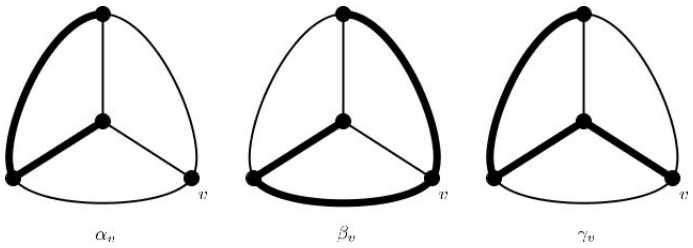


Fig. 2. The three canonical partial solutions. The three outer nodes are in S, while the central node is in A_S.

We are now ready to describe the algorithm that, given an SC 3-tree, determines an LCST if it has one. We highlight that, in order not to overburden the exposition, we focus on the decisional problem. It is not difficult, given the

information gathered in the decisional version of the algorithm, to find the edges of the LCST, as will be explained later.

From now on, we assume $\mathcal{L}_4(G)$ to be rooted in a degree 1 node, R. For any node $X \neq R$, we denote by $f(X)$ its parent, that is the first node encountered on the unique path from X to R; by $\bar{X}$ we denote the set of nodes of $\mathcal{L}_4(G)$ in the subtree rooted at X. To not clutter the exposition we will sometimes denote with $\bar{X}$ also the corresponding set of nodes in G, that is $\{x \in V(G) : x \in Y \text{ and } Y \in \bar{X}\}$.

Since we established the equivalence between minimal separators of G and edges of $\mathcal{L}_4(G)$ (cf. Lemma 4), in the following we will identify a minimal separator $S = X \cap f(X)$ of G with the corresponding edge $Xf(X)$ of $\mathcal{L}_4(G)$; moreover, we define the set of labels of edge $Xf(X)$ as $L(Xf(X)) = \{\chi_v : \exists \text{ a partial solution on } \overline{X} \text{ w.r.t. } X \cap f(X) \text{ with label } \chi_v\}$.

The very high level idea of the algorithm consists in traversing $\mathcal{L}_4(G)$ in post-order; when visiting a node X, we compute the set $L(Xf(X))$ of labels using the sets of labels of the children of X, $Y_1, ..., Y_c$, which have already been computed. This is done with the aim of extending the partial solutions of $\overline{Y_i}$, combining them in a partial solution of $\overline{X}$. It is clear that G contains an LCST if and only if $L(YR) \neq \emptyset$, where Y is the only child of root R.

We now focus on the issue of assigning to an edge $Xf(X)$ its set of labels $L(Xf(X))$. First of all, notice that if X is a leaf, then the partial solutions of $\overline{X}$ are exactly the nine canonical partial solutions, so in this case $L(Xf(X))$ contains all nine labels, that is $L(Xf(X)) = \{\chi_v | \chi \in \{\alpha, \beta, \gamma\}, v \in X \cap f(X)\}$. Otherwise, assume for example that X has two children Y_1, and Y_2. By brute force we test every pair of 2 labels, each one from the set $L(Y_1X) \times L(Y_2X)$, that is the cartesian product of $L(Y_1X)$ and $L(Y_2X)$. We "decode" these labels in the corresponding canonical partial solution, which we combine, together with a subset of edges E' of $E(X)$, in a subgraph H of $G[X \cup Y_1 \cup Y_2]$. If this subgraph is a partial solution of $G[X \cup Y_1 \cup Y_2 \cup f(X)]$ w.r.t. separator $X \cap f(X)$, then we add the corresponding label to $L(Xf(X))$; the following algorithm does the job, and its correctness is proved below.

Algorithm Compute-Labels
Input: An edge $Xf(X)$ of $\mathcal{L}_4(G)$;
Output: The set of labels $L(Xf(X))$

$G' \leftarrow G[X \cup f(X) \cup \bigcup_{i=1}^{c} Y_i]$;
$L(Xf(X)) \leftarrow \emptyset$;
foreach $(\chi^1_{v_1}, ..., \chi^c_{v_c}) \in L(Y_1X) \times ... \times L(Y_cX)$ **do**
  Let H_i be the canonical part. sol. of $G'[Y_i]$ associated to $\chi^i_{v_i}$, $i = 1, ..., c$;
  foreach subset $E' \subseteq E(X)$ **do**
    $H \leftarrow (X \cup \bigcup_{i=1}^{c} Y_i, E' \cup \bigcup_{i=1}^{c} E(H_i))$;
    if H is a partial solution of $G'[X \cup \bigcup_{i=1}^{c} Y_i]$ w.r.t. separator $Xf(X)$ **then**
      Add to $L(Xf(X))$ the label corresponding to H;
return $L(Xf(X))$.

Notice that, when $c = 0$, the cartesian product $L(Y_1X) \times ... \times L(Y_cX)$ is by definition equal to the set containing the empty tuple $\{()\}$; so, when X is a leaf, the external cycle is executed exactly once, the H_i's do not exist, and $H = (X, E')$ in every iteration of the inner cycle. Also, notice that when at least one of the $L(Y_iX)$'s is empty, then $L(Y_1X) \times ... \times L(Y_cX) = \emptyset$, so the outer cycle is never executed, and the output $L(Xf(X))$ is the empty set.

Before proving the correctness of Algorithm **Compute-Labels**, we highlight that its time complexity is constant, since the cardinality of $L(Y_1X) \times ... \times L(Y_cX)$ is always at most 9^3, there are a constant number of subsets of $E(X)$, and the "if" condition can be verified in constant time since G' has size $O(c) = O(1)$. Notice that **Compute-Labels** may be substituted by a constant size look-up table, if one is interested in the overall efficiency of the algorithm (see Figure 3 in Appendix).

The following lemma is needed in the proof of the correctness of algorithm **Compute-Labels**. Since it is an immediate consequence of the definition of partial solution, its proof is omitted.

Lemma 11. *Let G be a SC 3-tree, $Xf(X)$ be an edge of $\mathcal{L}_4(G)$. Let $G' = G[(V \setminus \overline{X}) \cup X]$. Assume G has a LCST T such that $T[\overline{X}]$ is a partial solution with label χ_v. Then, G' has a LCST T' such that $T'[X]$ is the canonical partial solution associated to label χ_v.*

Lemma 12. *After the execution of Algorithm* **Compute-Labels** *on input $Xf(X)$, $L(Xf(X))$ contains label χ_v, ($\chi \in \{\alpha, \beta, \gamma\}$ and $v \in X \cap f(X)$) if and only if there exists a partial solution of $G[\overline{X}]$ having label χ_v.*

Proof. see Appendix.

We are ready to give the pseudocode of the algorithm deciding whether an SC 3-tree has an LCST or not.

Algorithm Decide-LCSTonSC3-trees
Input: An n node SC 3-tree G;
Output: *yes* if an LCST of G exists, *no* otherwise.

if $n = 4$ **then return** *yes*;
Compute $\mathcal{L}_4(G)$;
Root $\mathcal{L}_4(G)$ in a degree 1 node R;
Let Y be the only child of R;
foreach node $X \neq R$ of $\mathcal{L}_4(G)$ in postorder **do**
    $L(Xf(X)) \leftarrow$ **Compute-Labels**$(Xf(X))$;
if $L(YR) \neq \emptyset$ **then return** *no*;
else return *yes*.

Theorem 4. *(Correctness and Complexity) Algorithm* **Decide-LCSTonSC3-trees** *returns "yes" if and only if the SC 3-tree in input has an LCST, in linear time.*

Proof. If $n = 4$, then obviously any spanning tree of G is locally connected, and the algorithm returns "yes". Otherwise, by Lemma 12, we have that $L(YR)$ is nonempty if and only if there exists a partial solution H of $G[\overline{Y}]$ with respect to the minimal separator YR. It is easy to see that H can be extended to a LCST of G adding a single edge.

Moreover, the algorithm is linear. Indeed, $\mathcal{L}_4(G)$ can be computed in linear time (cf. Lemma 5), and algorithm **Compute-Labels** is called $O(n)$ times, and, as already noted, has constant cost. □

In this extended abstract there is no space to detail how to reconstruct an LCST from the labels assigned in the algorithm, so here we will give only an overview. We can traverse again $\mathcal{L}_4(G)$, this time in a pre-order fashion; starting from the edge incident to the root, we arbitrarily choose one label; this label implies a certain canonical partial solution, so we add the corresponding edges to the current LCST. At the general iteration, we proceed visiting the children of the current node having already chosen a label of the separator corresponding to the edge connecting it with its father; this label came up from precise labels on the edges connecting this node to its children, so we are forced to choose exactly those labels, and we add in the LCST the corresponding edges of G.

5 Conclusions and Open Problems

We have proved that the problem of finding an LCST is linearly solvable on the classes of SC 3-trees; an analogous result holds for the case when the input graph is an SC 2-tree. Even supported by the results in [2], we conjecture that the LCST problem remains linear for every class of SC k-trees, for any k; this result may be achieved by giving a generalization of Theorem 3 to the class of SC k-trees, $k > 3$. This generalization requires to:

- extend Theorem 3, becoming: *Let G be a SC k-tree, and T be a spanning tree of G. T is locally connected if and only if for every minimal separator S of G, and for every $x \in S$:*
  $$N_T(x) \cap S = \emptyset \Rightarrow N_T(x) \subseteq A_S \text{ or } N_T(x) \subseteq B_S;$$
- increase the number of labels and specify their description;
- generalize Algorithm **Compute-Labels**.

A minor modification of this algorithm would allow the enumeration of all LCSTs of an SC k-tree, still running in polynomial time, for any fixed k. Indeed, the algorithm would have polynomial delay and the number of LCSTs in an SC k-tree is upper bounded by $nf(k)$, where f is the number of possible labels (f is exponential in k or worse). This follows from the fact that an SC k-tree has at most n minimal separators, and that an LCST can behave in at most $f(k)$ different ways on a minimal separator.

Moreover, we highlight that the result by Cai characterizes the SC 2-trees not admitting an LCST by means of a forbidden configuration, i.e. the 3-sun graph.

For what concerns the SC 3-trees, we know that some of them do not admit an LCST (for example the one whose $\mathcal{L}_4(G)$ has one node of degree 4 and three of the adjacent nodes have degree 4; all the other nodes are leaves); we wonder whether, also in this case, it is possible to characterize them by means of certain forbidden configurations.

References

1. J.S. Andrade, H.J. Herrmann, R.F.S. Andrade, L.R. da Silva, Apollonian Networks: Simultaneously Scale-Free, Small World, Euclidean, Space Filling, and with Matching Graphs *Phys. Rev. Lett.* 94, 2005. Erratum in *Phys. Rev. Lett.* 102, 2009.
2. S. Arnborg, and A. Proskurowski, Linear Time Algorithms for NP-Hard Problems Restricted to Partial k-trees. *Discrete Applied Mathematics* 23, 11–24, 1989.
3. L. Cai, On spanning 2-trees in a graph. *Discrete Applied Mathematics* 74, 203–216, 1997.
4. L. Cai, The complexity of the locally connected spanning tree problem. *Discrete Applied Mathematics* 131, 63–75, 2003.
5. M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980.
6. V.B. Le, Perfect k-line graphs and k-total graphs, *J. Graph Theory* 17, 65–73, 1993.
7. C.-C. Lin, G.J. Chang, G.-H. Chen, Locally connected spanning trees in strongly chordal graphs and proper circular-arc graphs. *Discrete Mathematics*, 307(2),208–215, 2007.
8. C.-C. Lin, G.-H. Chen, G.J. Chang, A linear-time algorithm for finding locally connected spanning trees on circular-arc graphs. *Algorithmica*, 66(2), 369–396, 2013.
9. L. Markenzon, C.M. Justel, and N. Paciornik, Subclasses of k-trees: Characterization and recognition. *Discrete Applied Mathematics* 154(5), 818–825, 2006.
10. B.S. Panda and D. Pradhan, Locally connected spanning trees in cographs, complements of bipartite graphs and doubly chordal graphs. *Information Processing Letters*, 110(23), 1067–1073, 2010.
11. A. Proskurowski, Separating Subgraphs in k-trees: Cables and Caterpillars. *Discrete Mathematics* 49, 275–285, 1984.
12. D.J. Rose, On simple characterizations of k-trees. *Discrete Mathematics*, 7, 317–322, 1974.
13. D.J. Rose, R.E. Tarjan, G. Luker, Algoritmic aspects of vertex elimination on graphs, *SIAM J. Comput. 5*, 266–283, 1976.
14. A. M. Farley, Networks immune to isolated failures, *Networks*, 11(3), 1981.

Appendix

Proof of Lemma 1.

Proof. Proceed by induction on the number n of nodes in $G = (V, E)$. If $n = k+1$, then G is a $(k+1)$-clique and the claim is trivially true because no separator exists. Assume now that the claim is true for each SC k-tree with less than $n > k+1$ nodes and let G be an SC k-tree with n nodes. In view of the recursive definition of the SC k-trees, there is a node t in G and a k-clique K such that $G' = G[V \setminus \{t\}]$ is an SC k-tree with $n-1$ nodes and K is the set of neighbors of t in G. Note that the separators of G are all the separators of G' plus the separator K. Let S be a separator of G. Two cases can arise.

- *S is the k-clique K.* In this case let $A_S = V \setminus (S \cup \{t\})$ and $B_S = \{t\}$ and the claim follows since $G[A_S \cup S]$ is the SC k-tree G' and $G[B_S \cup S]$ is the SC- k tree given by the k-clique K.
- *S is also a separator in G'.* By the inductive hypothesis there exist two sets A'_S and B'_S satisfying the claim in G'. Note that it cannot be both $A'_S \cap K \neq \emptyset$ and $B'_S \cap K \neq \emptyset$ (otherwise S would not be a separator). W.l.o.g. assume $B'_S \cap K = \emptyset$ and consider $A_S = A'_S \cup \{t\}$ and $B_S = B'_S$. Note that A_S is a connected component in G (since A'_S is a connected component in G' and t is connected to at least a node in A'_S). Moreover $G[A_S \cup S]$ is the SC k-tree obtained connecting the node t to the k-clique K in the SC k-tree $G'[A'_S \cup S]$ and $G[B_S \cup S]$ is the SC k-tree $G'[B'_S \cup S]$.

□

Proof of Lemma 3.

Proof. Proceed by induction on the number n of nodes in G. If $n = k+1$, then G is a $(k+1)$-clique and the claim is trivially true because no separator exists. Assume now that the claim is true for each SC k-tree with less than n nodes and let G be an SC k-tree with n nodes. In view of the recursive definition of the SC k-trees, there is a node a in G and a k-clique K such that $G' = G - \{a\}$ is an SC k-tree with $n-1$ nodes and K is the set of neighbors of a in G. Note that K is a (a, b)-minimal separator for G where b is a node in G' connected to all the nodes in K (the existence of such a node is ensured by item *(iv)* of Lemma 2). Thus the separator K of G satisfies the claim. Moreover all the others separators in G are also separators in G' thus the claim follows by inductive hypothesis. □

Proof of Lemma 4.

Proof. We prove the two implications separately.

($\Rightarrow$) Since S is a minimal separator in G, by Lemma 3 there are in G two nodes a and b such that S is an (a, b)-separator and $S \cup \{a\}$ and $S \cup \{b\}$ are $k+1$-cliques in G. Let X_1 and X_2 be the two nodes in $\mathcal{L}_{k+1}(G)$ corresponding to the $k+1$-cliques $S \cup \{a\}$ and to $S \cup \{b\}$ respectively. By definition of $\mathcal{L}_{k+1}(G)$, these nodes are adjacent and the claim follows.

($\Leftarrow$) We will show that the k-clique $S = X_1 \cap X_2$ in G is a minimal (a, b)-separator where $a = X_1 \setminus S$ and $b = X_2 \setminus S$.

Suppose, by contradiction, that a and b are connected in the subgraph G' induced by the nodes $V \setminus S$. Note that a and b not are adjacent in G (otherwise we have in G a $(k+2)$-clique induced by nodes of $X_1 \cup X_2$ against item *(i)* in Lemma 2) and let $P =< a, t_1, \ldots t_i, b >$, with $i \geq 1$, be a shortest path from a to b in G'. We will prove that each node of P must be connected to all the nodes in the set S, so leading to a contradiction since set $\{t_1\} \cup X_1$ forms a $(k+2)$-clique in G.

To prove that each node t_i in P must be adjacent to all the nodes in the set S, let us assume, by contradiction that there is a node t_j in P and a node u in S such that t_i and u are not adjacent. Let t be the first node adjacent to u we meet along the path P from t_j to a and let t' be the first node adjacent to u we meet along the path P from t_i to b. Now consider in G the cycle consisting of the nodes in P from t to t' and the node u. This cycle contains at least 4 nodes (i.e. t, t_j, t' and u) and is cordless (since P is a minimal path from a to b). Thus we have a contradiction in view of item *(iii)* in Lemma 2. □

Proof of Lemma 5.

Proof. A *perfect elimination ordering (peo)* of G is an order $v_1, v_2 \ldots v_n$ of its nodes such that the set $Pred(v_i)$, $1 \leq i \leq n$, of the nodes that are adjacent to v_i in G and that precede v_i in the order, form a clique.
Rose and al. in [13] developed a method based on Lexicographic Breadth First Search (*lex*-BFS that produces a peo for chordal graphs (and obviously for SC k-trees) in linear time. Moreover in [12] Rose proved that the peo produced with this method for k-tree (and obviously for SC k-trees) has the property that the first $k+1$ nodes in the order form a $k+1$-clique and $|Pred(v_i)| = k$ for each v_i, $k+1 < i <\leq n$. Once produced a peo $v_1, v_2 \ldots v_n$ with these properties it is easy to construct in linear time the tree $\mathcal{L}_{k+1}(G)$. We start with a node X_1 containing the nodes $x_1, x_2, \ldots x_{k+1}$. Moreover we obtain the other nodes of $\mathcal{L}_{k+1}(G)$ starting from the nodes v_i, $k+1 < i \leq n$. More precisely for v_i we create a node X_i containing the nodes $\{v_i\} \cup Pred(v_j)$, where v_j is the last predecessor of v_i, and we connect this new node X_i to the node X_j if $j > k+1$, to X_1 otherwise. It is easy to see that each of the $n-k$ nodes of the resulting tree is a $k+1$-clique and that, for each edge X_iX_j of the tree ,it holds $|X_i \cap X_j| = k$ (i.e. the tree is the the graph $\mathcal{L}_{k+1}(G)$). □

Proof of Lemma 6.

Proof. Let x be an isolated node in $T[S]$. By contradiction, assume that $N_T(x)$ has a non empty intersection both with A_S and with B_S. Let a and b be the T-neighbors of x such that $a \in A_S$ and $b \in B_S$; since T is an LCST, a and b must be connected in G by either an edge or a path not passing through any other node of S, and this is a contradiction since S is a separator.

In order to prove the second assertion, assume by contradiction that subgraph $G[S]$ does not contain any edge of T. By the first assertion, each node in S has its T-neighbors either all in A_S or all in B_S. But in this case, for each node $a \in A$ and $b \in B$ it cannot exists a path in T connecting a to b. Thus T is not a spanning tree. □

Proof of Lemma 7.

Proof. If G contains a 3-sun, then there are three 3-cliques all having an edge in common with the same (central) 3-clique, hence $\mathcal{L}_3(G)$ has a node of degree at least 3. Vice-versa let $X = \{x, y, z\}$ be a node of degree 3 in $\mathcal{L}_3(G)$. Let $Y_1 = \{x, y, a\}$, $Y_2 = \{x, z, b\}$ and $Y_3 = \{y, z, c\}$ be the three neighbors of X, then the six nodes $\{x, y, z, a, b, c\}$ induce a 3-sun in G. □

Proof of Lemma 8.

Proof. We prove the two implications separately.

($\Rightarrow$) If T is an LCST, the claim immediately follows from property (ii) in Lemma 6.

($\Leftarrow$) Proceed by induction on the number n of nodes in G. If $n = 3$ the claim trivially holds, because no separator exists. If $n > 3$ there exists at least a separator $S = \{x, y\}$. In view of Lemma 1 and by definition of k-paths, $G[A_S \cup S]$ and $G[B_S \cup S]$ are 2-paths and hence they satisfy the inductive hypothesis, implying that $T[A_S \cup S]$ and $T[B_S \cup S]$ are LCSTs. Merging together $T[A_S \cup S]$ and $T[B_S \cup S]$ we get a tree T that is a LCST because edge xy belongs to T. □

Proof of Lemma 9.

Proof. Suppose, by contradiction, that there exists a 4-clique $X = \{x, y, z, t\}$ in $G = (V, E)$ such that $T[X] = 2K_2$. This implies that, for each minimal separator $S \subset X$ of G, it holds (ii). Let $S = \{x, y, z\}$ be any of these separators and let z be the isolated node in $T[S]$. Without loss of generality let B_S be the component of $G[V \setminus S]$ containing $N_T(z)$. In graph $T[A_S \cup S]$ node z is hence isolated. The above reasoning applies to any other separator in X. Thus the path in T connecting the two edges of the $2K_2$ must be in X and this contradicts the assumption that $T[X] = 2K_2$. □

Proof of Lemma 10.

Proof. First we show that G is a 3-path. By contradiction assume that tree $\mathcal{L}_4(G)$ is not a path. Let $X = \{x, y, z, t\}$ be a node of $\mathcal{L}_4(G)$ with three neighbors Y_1, Y_2 and Y_3. Consider now the three minimal separators $S_i = X \cap Y_i$, $1 \leq i \leq 3$ (cf. Lemma 4). In order to fix the ideas let $t \in X$ be the node in $\bigcap_i Y_i$ and $S_1 = \{x, y, t\}$, $S_2 = \{x, z, t\}$ and $S_3 = \{z, y, t\}$. In each graph $T[S_i]$, $1 \leq i \leq 3$, there is a single isolated node, if this node is t for all the three graphs, then xy,

xz and yz are in T. Thus the set $\{x, y, z\}$ is a cycle in T, a contradiction. Hence there is a minimal separator S_i,$1 \leq i \leq 3$, such that t is not isolated in $T[S_i]$, w.l.o.g. let it be S_1 and let $xt \in E(T)$. This implies that y is the only isolated node in $T[S_1]$, z is the only isolated node in $T[S_2]$ and the edges xy, ty, xz and tz are not in T. This in turn implies that yz is the only edge in $T[S_3]$. Summarizing, we have that the edges xt and yz form a $2K$ in the 4-clique X, a contradiction to Lemma 9.

Now we show that T is an LCST. We proceed by induction on the number n of nodes in the 3-path $G = (V, E)$. If $n = 4$, then G is a 4-clique and the claim is trivially true because any spanning tree of a 4-clique is locally connected. Assume now that the claim is true for every 3-path with less than $n > 4$ nodes and let G be a 3-path with n nodes. Let t be the last node added to G in its recursive definition and consider the separator $S = \{x, y, z\}$ identified by the neighbors of t in G. Observe that $G' = G[V \setminus \{t\}]$ is a 3-path with $n - 1$ nodes since $\mathcal{L}_4(G')$ can be obtained by $\mathcal{L}_4(G)$ by cutting the leaf containing t. Moreover, for all the separators of G' (note that these separators are also separators of G) $T' = T[V \setminus \{t\}]$ satisfies (ii) in G'. Since T satisfies (ii) for the separator S in G, t cannot be adjacent in T to both x and y (otherwise a cycle is introduced in a tree), so the degree of t in T is at most 2. We will examine the two cases.

1. $|N_T(t)| = 1$: T' is a spanning tree for G' and, by inductive hypothesis, it is an LCST of G'. Moreover, since T is connected it must be either $xt \in T$ or $yt \in T$ (remember that S satisfies (ii)). W.l.o.g. assume that $xt \in T$. For each $u \in V$, it holds

$$N_T(u) = \begin{cases} \{x\} & \text{if } u = t \\ N_{T'}(x) \cup \{t\} & \text{if } u = x \\ N_{T'}(u) & \text{otherwise} \end{cases}$$

   Now, using the fact that T' is an LNCS for G' and that $yt \in E$ and $y \in N_{T'}(x)$, it is easy to see that T is an LCST for G.
2. $|N_T(t)| = 2$: Without loss of generality assume that $N_T(t) = \{x, z\}$. Note that $N_T(z) = \{t\}$ since S satisfies (ii) in T. Let a be a node in G connected to all nodes in S (such a node there exists by Lemma 3). Since G is a 3-path, any induced 4-clique (in particular, $K = \{a, x, y, z\}$) contains at most 2 minimal separators, so besides $\{x, y, z\}$, at most one among $\{x, y, a\}$, $\{x, z, a\}$ and $\{y, z, a\}$ is a minimal separator contained in K. There are three cases to consider.
   (a) *S is the only minimal separator in K.* In this case G has only five nodes (the nodes $\{a, x, y, z, t\}$). The node a in T can be connected only to x or to y.
   In the first case (where ax is in T), we have $N_T(x) = \{y, t, a\}$ and $N_T(t) = \{x, z\}$. Moreover x and t are the only nodes in T having degree greater than one. Thus we conclude that T is an LCST by noting that ay, yt and xz are edges of G.
   In the second case (where ay is in T), x, t and y are the only nodes in T having degree greater then one and $N_T(x) = \{y, t\}$, $N_T(t) = \{x, z\}$ and

$N_T(y) = \{x, a\}$. Thus again T is an LCST by noting that yt, xz and xa are edges of G.

(b) $S' = \{a, x, y\}$ *is a minimal separator in* K. $A_{S'}$ contains only t and z and $T' = T[B_{S'} \cup S']$ is a spanning tree in $G' = G[B_{S'} \cup S']$ that satisfies (ii) on every minimal separator, hence -by inductive hypothesis - G' is a 3-path and T' is an LCST. Summarizing, for each $u \in V$, it holds

$$N_T(u) = \begin{cases} \{z, x\} & \text{if } u = t \\ \{t\} & \text{if } u = z \\ N_{T'}(x) \cup \{t\} & \text{if } u = x \\ N_{T'}(u) & \text{otherwise} \end{cases}$$

Now, using that T' is an LCST of G', $zx \in E$, $y \in N_{T'}(x)$ and $yt \in E$, it is easy to see that T is a LCST of G.

(c) S' *is a minimal separator in* K *and either* $S' = \{a, y, z\}$ *or* $S' = \{a, x, z\}$. Assume first that $S' = \{a, y, z\}$. Note that T' obtained by adding edge xz to $T[V \setminus \{t\}]$ is a spanning tree of $G' = G[V \setminus \{t\}]$ and it satisfies (ii). Thus, by inductive hypothesis, T' is an LCST of G'. Note that S' satisfies (ii) on T and since $N_T(z) = \{t\}$ it must be $ay \in E(T)$. Summarizing, for each $u \in V$, it holds

$$N_T(u) = \begin{cases} \{z, x\} & \text{if } u = t \\ \{t\} & \text{if } u = z \\ \{y, t\} & \text{if } u = x \\ N_{T'}(u) & \text{otherwise} \end{cases}$$

Now, using that T' is an LCST of G', and zx and yt are edges of G, it is easy to see that T is an LCST of G.

The reasoning is similar if we assume that the minimal separator S' is $\{a, x, z\}$. In this case we can consider the spanning tree T' of G' obtained by adding the edge yz to $T[V \setminus \{t\}]$.

□

Proof of Lemma 12.

Proof. ($\Rightarrow$) If X is a leaf, that is $c = 0$, then the canonical partial solution on $G[\overline{X}] = G[X]$ corresponding to label χ_v satisfies the statement.

Else, assuming $1 \leq c \leq 3$, let E' be the selected subset of $E(X)$ such that H is a partial solution of $G'[X \cup \bigcup_{i=1}^{c} Y_i]$ w.r.t. separator $Xf(X)$, for which label χ_v was added to $L(Xf(X))$. By structural induction, there are partial solutions $\overline{H}_i$ of $G[\overline{Y}_i]$, having label $\chi^i_{v_i}$, $i = 1, ..., c$. By Lemma 11, if $(\overline{X}, E' \cup \bigcup_{i=1}^{c} E(\overline{H}_i))$ was not a partial solution of $G[\overline{X}]$, then neither would H be a partial solution of $G'[X \cup \bigcup_{i=1}^{c} Y_i]$, a contradiction. Finally, notice that $G[\overline{X}]$ has the same label of H, that is χ_v.

($\Leftarrow$) If X is a leaf, then $G[\overline{X}] = G[X]$ admits all nine possible canonical partial solutions. Since $c = 0$, the outer cycle is executed exactly once on the empty tuple (), and the inner cycle will find a partial solution for each possible

label. Assume now that $1 \leq c \leq 3$. Assume there exists a partial solution $\overline{H}$ of $G[\overline{X}]$ with respect to separator $Xf(X)$. Then, notice that $\overline{H}[\overline{Y}_i]$ is a partial solution of $G[\overline{Y}_i]$ with respect to separator Y_iX with label $\chi^i_{v_i}$, so by structural induction $L(Y_iX)$ contains label $\chi^i_{v_i}$, $i = 1, ..., c$. Let H_i be the canonical partial solution of $G[Y_i]$ with respect to separator Y_iX. Then, by Lemma 11, $H = (X \cup \bigcup_{i=1}^{c} Y_i, E' \cup \bigcup_{i=1}^{c} E(H_i))$ is a partial solution of $G'[X \cup \bigcup_{i=1}^{c} Y_i]$ with respect to separator $Xf(X)$, where $E' = E(\overline{H}[X])$ is found by brute force by the inner cycle, and the corresponding label (that is the same as $\overline{H}$) is added to $L(Xf(X))$. □

Table used in Algorithm Decide-LCSTonSC3-trees

label of $\{x,y,t\}$	label of $\{x,z,t\}$	label of $\{y,z,t\}$	label of $\{x,y,z\}$
α_x	-	-	$\alpha_x(+yz), \beta_z(+xz,yz), \gamma_y(+zt,xz)$
α_y	-	-	$\alpha_y(+xz), \beta_z(+xz,yz), \gamma_x(+yz)$
α_t	-	-	$\beta_x(+xz,zt), \beta_y(+yz,zt)$
β_x	-	-	$\alpha_z, \beta_x(+xz), \beta_y(+yz), \gamma_z(+tz)$
β_y	-	-	$\alpha_z, \beta_x(+xz), \beta_y(+yz), \gamma_z(+tz)$
β_t	-	-	$\gamma_y(+xz), \gamma_x(+yz)$
γ_x	-	-	$\gamma_y(+xz), \gamma_x(+yx)$
γ_y	-	-	$\gamma_y(+xz), \gamma_x(+yz),$
γ_t	-	-	$\alpha_z, \beta_x(+xz), \beta_y(+yz)$
α_x	α_t	-	$\beta_z(+yz)$
α_x	β_z	-	γ_y
α_t	α_x	-	$\beta_y(+yz)$
α_t	β_z	-	β_x
α_t	γ_t	-	β_x
β_x	α_z	-	$\alpha_z, \beta_y(+yz)$
β_x	γ_z	-	γ_z
β_x	β_x	-	β_x
β_x	β_t	-	γ_z
β_y	α_x	-	γ_z
β_y	α_t	-	β_x
β_t	α_z	-	$\gamma_x(+yz)$
β_t	β_x	-	γ_y
γ_y	β_x	-	γ_y
γ_t	α_t	-	β_x
α_x	α_t	β_y	β_z
α_x	β_z	β_t	γ_y
α_t	α_x	β_z	β_y
α_t	β_z	α_y	β_x
β_x	α_z	α_t	β_y
β_x	β_t	α_y	γ_z
β_y	α_x	β_t	γ_z
β_y	α_t	α_z	β_x
β_t	α_z	β_y	γ_x
β_t	β_x	α_z	γ_y

Fig. 3. Table for the construction of the labels. Symbol '-' means that the corresponding child is not present in $\mathcal{L}_4$. Notice that missing combinations do not lead to any feasible label for separator $\{x,y,z\}$.

Gathering of Robots in a Ring with Mobile Faults

Shantanu Das[1], Riccardo Focardi[2], Flaminia L. Luccio[2], Euripides Markou[3], Davide Moro[2], and Marco Squarcina[2]

[1] Aix Marseille Univ, CNRS, LIF, Marseille, France
[2] DAIS, Università Ca' Foscari Venezia, Venezia, Italy
[3] DCSBI, University of Thessaly, Lamia, Greece

Abstract. In this paper, we consider the problem of gathering mobile agents in a graph in the presence of mobile faults that can appear anywhere in the graph. Faults are modeled as a malicious mobile agent that attempts to block the path of the honest agents and prevents them from gathering. The problem has been previously studied by a subset of the authors for asynchronous agents in the ring and in the grid graphs. Here, we consider synchronous agents and we present new algorithms for the unoriented ring graphs that solve strictly more cases than the ones solvable with asynchronous agents. We also show that previously known solutions for asynchronous agents in the oriented ring can be improved when agents are synchronous. We finally provide a proof-of-concept implementation of the synchronous algorithms using real Lego Mindstorms EV3 robots.

1 Introduction

An important problem in the area of distributed computing with mobile agents (or robots) is the *rendezvous* problem, i.e., the gathering of all agents at the same location. Gathering is a crucial task for teams of autonomous mobile robots, since they may need to meet in order to share information or coordinate. This problem has been widely studied when the environment is modelled as an undirected graph and the mobile agents can move along the edges of the graph. Most of the studies are restricted to fault-free environments and very little is known about gathering in the presence of faults. Possible faults can be a permanent failure of a node, like for example the so called *black hole* that destroys agents arriving at a node, or, transient faults that can appear anywhere in the graph, such as a mobile hostile entity (an *intruder*) that behaves maliciously. Some work has been done to locate a malicious node in a graph (see, e.g., [8,10]), whereas protecting the network against a malicious entity that is mobile and moves from node to

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 122–135 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

node of the graph, is a more difficult problem. An example of this kind is the so-called *network decontamination* or *intruder capture* problem (see, e.g., [9,11]).

In this paper we consider a new type of *malicious agent* that was introduced in [5]. This agent can block the movement of an honest agent to the node it occupies, it can move arbitrarily fast along the edges of the graph, it has full information about the graph and the location of the agents, and it even has full knowledge of the actions that will be taken by the honest agents. On the other hand, the honest agents are relatively weak: they are independent and identical anonymous processes with some internal memory, they move synchronously and use face to face communication when they are in the same node. We investigate the feasibility of rendezvous in oriented and unoriented ring networks in the presence of such a powerful adversary, that blocks other agents from having access to parts of the network.

We consider an oriented and an unoriented ring network modelled as a connected undirected graph with a set of mobile honest agents located at distinct nodes of the graph, and a malicious agent that cannot harm the honest agents but can prevent them from visiting a node. As in [5] we are interested in solving the rendezvous of all honest agents in this hostile environment. However, differently from [5], we consider synchronous mobile honest agents. These agents have local communication capability, and no prior knowledge of their number k (except for the protocol for $k = 2$) and of the ring size n. Our objective is to study the feasibility of rendezvous with minimal assumptions.

Contributions. (*i*) We prove that in an unoriented ring network of n nodes the problem is not solvable for n odd and k even, and we then present an algorithm that solves the problem in all the other cases for $k > 2$ synchronous agents, also in the case n odd and k odd which is unsolvable with asynchronous agents; (*ii*) we briefly discuss how to solve the problem for $k = 2$ agents and n even, not solvable in the asynchronous setting, and how to solve the problem in the oriented ring by improving the known algorithm for asynchronous agents; (*iii*) we show that the proposed algorithms and underlying assumptions are realistic through a proof-of-concept implementation using Lego Mindstorms EV3 robots.

Related work. In this paper we consider the rendezvous problem, which has been previously studied for robots moving on the plane [3], or for agents moving on graphs [1]. This problem can be easily solved, in synchronous and asynchronous settings, when the system is not symmetric, e.g., when there is a distinguished node. In symmetric settings (e.g., in the anonymous ring) the problem is non-trivial and symmetry has to be broken, e.g., by using tokens [6], by adding distinct identifiers to agents [4], etc.

Some recent work [7] also considers the problem of rendezvous in the presence of *Byzantine agents*, which are indistinguishable from the *honest* agents, but may behave in an arbitrary manner, and may also provide false information to the honest agents so to prevent their gathering. Other related work includes gathering in networks with delay faults [2]. The model used in [2] is somehow similar to ours in the sense that the adversary may delay for an arbitrary, but finite time, the

movements of the agents. However, contrary to the model of [2], in our setting, one of the agents may be blocked forever.

From a practical point of view, the work that is more closely related to our model is [13], where the author proposes an example of a simulation of a known distributed fault-free gathering algorithm, using real NXT robots. Differently from our setting, in [13] robots are moving on a plane (not in a graph), do not use direct communication, but only indirect one, i.e., can only detect the position and the movement of the other robots but cannot communicate any information.

Paper structure. The paper is organized as follows: In Section 2 we illustrate the formal model; in Section 3 we first present some impossibility results, and then the algorithm for the unoriented ring network for $k > 2$ agents. For lack of space we only discuss the solution for $k = 2$ and for the oriented ring network. In Section 4 we present the implementation of the algorithms on programmable robots, and we discuss the usefulness of the study of the real model in the development of the theoretical models. We conclude in Section 5.

2 A Model of Synchronous Agents in a Ring

We consider a distributed network modeled by an undirected, connected graph $G = (V, E)$ where V are the anonymous and identical nodes or *hosts*, and E the edges or connections between nodes. We assume that in the network there are k *honest* anonymous identical synchronous agents $A_1, A_2, \ldots A_k$, and one *malicious* agent M which is trying to prevent the gathering of honest agents. More precisely, the network, and the capabilities and behavior of honest and malicious agents are the following.

The network: We consider either oriented or unoriented ring networks $R = (V, E)$, with $|V| = n$ nodes. The links incident to a host are distinctly labeled. In the oriented ring, the links are labelled consistently to allow all agents agree on a common sense of direction. In the unoriented ring there is no such agreement and the labels on the links are arbitrary. The edges of the network are FIFO links, meaning that all agents, including the malicious one, that move in the same link respect a FIFO ordering. We call a node v *occupied* when one or more *honest* agents are in v, and we call v *free* or *unoccupied* otherwise. Moreover, for solvability reasons, we will have to assume that the ring contains a special node marked $\otimes$, otherwise gathering is impossible (see [5]).

Honest agents: The agents are independent and identical, anonymous processes with some internal constant memory, except for the case $k = 2$ in the unoriented ring that requires $O(\log n)$ bits. These agents are initially scattered in the graph, i.e., at most one agent at a node, start the protocol at the same time, and can move along the edges of G. An agent located at a node v can see how many other agents are at v, and it can access their states. An agent cannot see or communicate with any agent that is not located at the same node, i.e., communication is face to face. Moreover an agent cannot mark the node or leave any information on the node that it visits. An agent arriving at a node v, learns the label of the incoming

port and the label of the outgoing port. Two agents traveling on the same edge in different directions do not notice each other, and cannot meet on the edge. Their goal is to rendezvous at a node. The agents neither have knowledge of the size n of the ring, nor of the number k of agents present in the network.

The moves of honest agents are synchronized as follows: (a) all agents start executing the protocols at the same time; (b) time is discretized into atomic time units; (c) all agents start in the same initial state, and the set of agent states is finite and independent of the graph size or the number of agents; (d) During each time unit, an agent arriving at a node v through a port q takes the following three actions: ($d.1$) it reads its own state, counts the number of agents at v and reads their states; ($d.2$) based on the above information it performs some computation to decide its next destination; the output of the computation is a new state and the port number p of an edge incident to v, or $p = 0$ if the agent decides to stay in v; ($d.3$) the agent changes its state and either moves using the computed port number ($p > 0$), or waits at the current node ($p = 0$). If the agent decided to move on edge (v, z), and the node z is not occupied by the malicious agent then the agent is located at the node z in the next time unit. Otherwise, the agent is still located at node v in the next time unit, with a flag set in its memory notifying the agent that the move was unsuccessful. In the next time unit, the agent repeats the above three actions.

Malicious agent: We consider a worst case scenario in which the malicious agent M is a very powerful entity compared to honest agents: It has unlimited memory; at any time has full knowledge of the graph, the locations of the honest agents and their states, and the algorithm followed by the agents. Based on the above information, M can either stay at its current node y or decide to move to another node w, if there is a path from y to w, such that no node on this path, including w, is occupied by any honest agent. The speed of the malicious agent is unbounded, thus M can move arbitrarily fast inside the network, but must move along the edges of the graph, and it also obeys the FIFO property of the links. When it resides at a node u it prevents any honest agent A from visiting u, i.e., it "blocks" A: If an agent A attempts to visit u, the agent receives a signal that M is in u. M can neither visit a node which is already occupied by some honest agent, nor cross some honest agent in a link.

3 Rendezvous in Rings with Synchronous Agents

In this section we first study the rendezvous problem with synchronous agents having constant memory, in an unoriented ring with one malicious agent. In [5] the authors have analyzed the same problem with asynchronous agents and have proved that the problem is solvable in any unoriented ring if and only if the number of honest agents k is odd. We prove here that with $k > 2$ synchronous agents having constant memory, the problem can be solved in an unoriented ring consisting of n nodes, when n is even or k is odd. We also prove that for k even and n odd numbers, the problem is unsolvable. We discuss also the case $k = 2$ in unoriented rings, and the problem in oriented rings.

Let us first show the impossibility result in the unoriented ring. The technique of the proof is similar to the one presented in Lemma 5 of [5], where it was proved that an even number of agents with constant memory cannot gather in an unoriented ring with a malicious agent. We can show that when n is odd and k is even, an adversary can initially place the agents in a configuration that is symmetric with respect to a line passing through the special node $\otimes$, and can place itself in $\otimes$. Hence, the agents are distributed into two groups of equal size. Moreover, the agents belonging to the same group, agree on the clockwise (CW) orientation but they do not agree with the agents of the other group. In other words, what is considered clockwise direction for any agent of one group is considered counter-clockwise direction for every agent of the other group. Each two symmetrically placed agents which belong to different groups should behave identically under any algorithm. Moreover, recall that $n-1$, the number of free positions is even, i.e., there is no meeting point in the middle opposite to $\otimes$, and therefore the agents of the two groups can never meet at a node. We thus have:

Lemma 1 (Impossibility when n is odd and k is even). *In an unoriented ring with a specially marked node $\otimes$ and a malicious agent having arbitrary speed, $k \geq 2$ mobile synchronous agents starting from arbitrary symmetric locations, cannot gather at a node if n is odd and k is even.*

3.1 Unoriented Ring with more than two Agents

We now propose an algorithm for $k > 2$ synchronous agents with constant memory, and that do not know n and k. Moreover, agents do not agree on a common sense of direction given that the ring is unoriented. Algorithm 1, called *CollectAgents*, works as follows. Each agent has a state and some local variables that keep track of the execution. Agents wake up in state *INITIAL*. We illustrate the algorithm distinguishing the case in which all agents have the same orientation, from the case in which the agents are split in two groups, those that move in one direction and those that move in the opposite direction. Note that the agents are not aware of this information so they cannot adapt their behaviour accordingly.

If an agent continues moving in the same direction and eventually reaches $\otimes$ for the second time, then the agent can be sure that all agents are moving in the same direction. In this case, one of the agents - the first agent to reach $\otimes$ twice - will stop at $\otimes$. The other agents continue moving in the same direction until they are blocked by M. The first agent to be blocked stops (in state *STOPPED*) and all other agents gather at this node, except one of the agents (the first one to meet the *STOPPED* agent) which changes to state *MSG* (“messenger") and goes around the ring in the other direction until it is blocked by M. At this point M is surrounded by the stopped agent (now in state *HEAD*) and the messenger agent. The messenger agent (changing to state *R_MSG*) returns back, collecting the *STAR* agent on the way, and rendezvous is achieved when these two agents reach the other agents who are waiting at the *HEAD* agent.

When the agents do not have the same orientation then the situation is different. In this case there are two groups of agents, G_1 and G_2, moving in

opposite directions. One agent from each group will eventually be blocked by M and become the "leader" (*HEAD*) agent for their group. The two groups of agents will start gathering in two distinct locations. To bring the two groups together, we use the fact that either k is odd or n is even. In the former case, one of the groups is of odd size, and agents of this group turn back and go to the other group. Otherwise if k is even and both groups are of even size, then the leader (*HEAD*) agents of the two groups try to push M from two sides until M is cornered in one node of the network. The segment of the ring containing the other nodes is of odd size and the two groups of agents can meet at the center of this segment. Let agent A for G_1, and agent B for G_2, be the agents that are blocked by M and become *STOPPED* (line 6). When two other agents, say C and D, meet A and B respectively then A and B will become *HEAD* and will start moving, trying to surround M, while C and D become *MSG*, reverse direction and move towards B and A, respectively. When agents C and D meet the two *HEAD* agents B and A (line 67), they reverse direction and become *R_MSG* delivering some information and joining all the agents of their own group. If either G_1 or G_2 is of odd size and all the agents of the odd group move towards even group (line 23), and rendezvous is achieved. If k is even, and both G_1 and G_2 are of even size, then agents C and D go back and forth, until when they find that M has been surrounded, i.e., when *HEAD* agents cannot move anymore (line 24). At this point G_1 and G_2 move towards the middle point and gather there if n is even or the algorithm fails if n is odd (line 38).

Algorithm 1 *CollectAgents*

```
#  Gathering k > 2 agents in an Unoriented Ring.
#  Local variables: sync = parity = s_reached = 0, moved = -1, first_clock = 1,
State = INITIAL

 1: sync := (sync + 1) mod 2

 2: if State = INITIAL then
 3:     dir := CW
 4:     if first_clock = 1 and Reached ⊗ then s_reached = s_reached +1
 5:     end if
 6:     if Blocked then State := STOPPED
 7:     else
 8:         Move to the next node
 9:         if Reached ⊗ then s_reached = s_reached +1
10:         end if
11:         if meet a STOPPED agent  then State := MSG
12:         else if meet a HEAD agent h then State := FOLLOWER of h
13:         else if Blocked then State := STOPPED
14:         else if s_reached = 2 and not found a STAR agent then
15:             State := STAR
16:         end if
17:     end if

18: else if State = HEAD then
19:     if meet a INITIAL agent  then parity:= (parity + 1) mod 2
20:     else if meet a R_MSG agent r then
21:         if r.moved = -1 then State := RENDEZVOUS
22:         else if parity = 1 and r.parity = 0 then  State := WAIT
23:         else if parity = 0 and r.parity = 1 then State := R_HEAD
24:         else if moved = 0 and r.moved = 0 then
25:             State := R_HEAD
26:             Wait (sync)
27:         end if
28:         moved:= 0
29:     else if not blocked and s_reached < 2 then
30:         moved:= 1
31:         Move to the next node
32:         if Reached ⊗ then s_reached = s_reached +1
33:         end if
34:     else Wait (1)
35:     end if

36: else if State = R_HEAD then
37:     dir:= CCW
38:     if Blocked then State := FAILURE
39:     else if meet a R_HEAD agent  then State := RENDEZVOUS
40:     else
41:         Move to the next node
42:         if meet a R_HEAD or WAIT agent  then State := RENDEZVOUS
43:         end if
44:     end if
```

CollectAgents algorithm - Continue

```
45: else if State = STOPPED then
46:     if meet a INITIAL agent  then
47:         State := HEAD
48:         moved:= 0
49:         parity:= (parity + 1) mod 2
50:     else if meet a MSG agent m then State := FOLLOWER of m
51:     else Wait (1)
52:     end if

53: else if State = STAR then
54:     if meet a R_MSG agent r then State := FOLLOWER of r
55:     else Wait (1)
56:     end if

57: else if State = FOLLOWER then
58:     if leader state is RENDEZVOUS then State := RENDEZVOUS
59:     else follow the leader
60:     end if

61: else if State = MSG then
62:     dir:= CCW
63:     if Blocked then
64:         State := R_MSG
65:     else
66:         Move to the next node
67:         if meet a HEAD agent h then
68:             state := R_MSG
69:             moved:= h.moved
70:             parity:= h.parity
71:         else if Blocked or meet a STOPPED agent  then state := R_MSG
72:         end if
73:     end if

74: else if State = R_MSG then
75:     dir:= CW
76:     Move to the next node
77:     if meet a HEAD agent h then
78:         if moved = -1 then State := RENDEZVOUS
79:         else if (parity!= h.parity) or (moved = 0 and h.moved = 0) then
80:             State := FOLLOWER
81:         else State := MSG
82:         end if
83:     end if

84: else if State = WAIT then
85:     if meet a R_HEAD agent  then
86:         State := RENDEZVOUS
87:         Change state of FOLLOWER agents to RENDEZVOUS
88:     else Wait (1)
89:     end if
90: end if
91: first_clock:= 0
```

Theorem 1. *Algorithm CollectAgents solves the rendezvous problem for $k > 2$ agents in an unoriented ring of size n with a special node $\otimes$, and with one malicious agent M. It returns a failure message when n is odd, k is even and there are at least two agents for each of the two possible orientations.*

Proof. All the agents start moving at the same time in their *CW* direction, but there is no common sense of direction thus some agents could move in one direction while some others are moving in the opposite one. We have three possible cases: (*1*) all the agents move in the same direction; (*2*) all the agents except one move in the same direction; (*3*) at least two agents move in one direction and at least two move in the opposite one.

(*1*) We want to prove that in this case exactly one agent becomes *STOPPED*. Let us first assume by contradiction that no agent becomes *STOPPED*. There are two cases: a) Either M stops or moves in a direction opposite to the other agents, but in this case it blocks one agent that becomes *STOPPED*, thus a contradiction; b) or M keeps on moving in the same direction chosen by the agents, but in this case one agent will cross $\otimes$ twice and will stop, thus blocking M. Therefore, at least another agent will eventually be blocked by M (no other agent stops at $\otimes$), and will become *STOPPED*, thus a contradiction also in this case. Let us now prove that the *STOPPED* agent is unique. Given that by hypothesis there are $k > 2$ agents that move in the same direction, and at most one agent becomes *STAR*, then one agent C has to meet A, the *STOPPED* one, becomes *MSG*, while A changes its state to *HEAD*. Note that, all the other agents in state *INITIAL* cannot become *STOPPED*, but become *FOLLOWER*, given that they meet A before being blocked by M. The agents A and C move in opposite directions. C reaches M in the opposite side because there are no other *STOPPED* or *HEAD* agents, given that C is the first agent that moves in this direction. Thus, the *STOPPED* agent is A and is unique. Let us now prove that C will coordinate the gathering. When C comes back in state *R_MSG* has the default *moved* value equal to -1. Agent A in the opposite side continues to move in the same direction, with some agents in state *FOLLOWER*. However, the two agents C and A eventually meet, in fact either A is stopped by M, or after it has reached $\otimes$ for the second time, it will remain stopped, together with the agents in state *FOLLOWER*, waiting for the arrival of C to rendezvous. Note that, if one agent had previously stopped in $\otimes$, then it has to be in the path followed by *R_MSG*, thus it will become a *FOLLOWER* of *R_MSG* and will rendezvous with the other agents. Thus, if all agents initially move in the same direction rendezvous is achieved.

(*2*) Let us assume that all the agents except one, called B, move in the same direction. The agent M cannot escape both from B and from the agents going in the opposite direction, let us call this group G_1, thus M has to block both B and another agent A of G_1. B becomes *STOPPED* but not *HEAD* because no other agent can reach it in state *INITIAL*, whereas A, becomes *STOPPED*, and once it is reached by an *INITIAL* agent C, A becomes *HEAD*, and C starts moving in the opposite direction, i.e. towards B, in state *MSG*. All the remaining agents of G_1 become *FOLLOWER* of A. When C and B meet, B becomes a *FOLLOWER*

of C, and C moves in state *R_MSG* towards A with the default *moved* value equal to -1. The agent A and its *FOLLOWER* agents continue to move in CW direction, however A is either blocked by M, or it stops after reaching $\otimes$ twice. When C and B reach A, all the other possible agents already reached A, and since $moved = -1$ the agents terminate in state *RENDEZVOUS*.

(*3*) At least two agents have an orientation in one direction, and another two in the other direction. Thus, exactly 2 agents become *HEAD* and 2 agents become *MSG*. Let us call them A and C - the *HEAD* and the *MSG* on one side, and B and D, those on the other side, respectively. All the other agents become *FOLLOWER* of the *HEAD* they meet. C and D perform a tour in opposite directions: they reverses direction, reach B and A, respectively, and come back to A and B in state *R_MSG*, delivering the *moved* and the *parity* information related to the other *HEAD* agent.
We now have to show that the two groups of agents are able to meet. First observe that since agents are synchronous and links are FIFO D (C) meets A (B) in state *MSG* before that C (D) comes back in state *R_MSG* delivering the information on the parity or oddness (computed using the *parity* bit) of the opposite group of agents, and the information on the possible movement of the opposite *HEAD*, stored in a variable *moved*. Note that, when M is surrounded, A and B are not able to move. We have two cases:

- k is odd: If k is odd, the *HEAD* agent of a group with $parity = 1$, remains stopped in state *WAIT*, whereas the other *HEAD* agent, with $parity = 0$, starts moving in opposite direction in state *R_HEAD* with all the other agents of its group in state *FOLLOWER*. This group of agents will reach the other stopped group -regardless of the value n- and all agents will rendezvous.

- k is even: This case is more complicated and rendezvous can be achieved only when M has been surrounded, i.e., when A and B cannot move anymore. This is possible since even if M is very fast, due to the synchronicity of the agents, M can block only A or B in one time step. Note also that none of the *HEAD* agents can meet $\otimes$ twice since M is surrounded before either of them could visit all the nodes of the ring.
Thus, we have now to prove that C and D perform a same number of turns and they eventually meet, together with their *FOLLOWER* agents. Given that C and D collect both the *moved* value of their *HEAD* and the one of the opposite *HEAD* agent, if at least one of the *moved* values is not 0 (which means at least one of the *HEAD* agents moved) they start a new round, otherwise they try to rendezvous. Observe that collecting both values and having FIFO links implies a synchronization of the rounds.
Let G_1 and G_2 be the two groups of agents surrounding M that will eventually be formed at the node of A and of B respectively. The *sync* variable changes at each clock (line 1), and is used to synchronize the movements. In case n is even, A and B are separated by an odd number of nodes. If one *HEAD* agent changes direction at an even clock tick and the other at an odd clock tick (defined by the *sync* variable), only one of the groups waits one clock cycle (line 26), and so the two groups move at an instant of time that preserves the odd distance and thus,

they rendezvous. Without this synchronization, the agents may cross on an edge without meeting.
Conversely, if n is odd A and B do not meet along the path and are blocked again by M, thus they move to the state *FAILURE* since they did not rendezvous. □

The complexity of the algorithm is as follows.

Theorem 2. *Algorithm CollectAgents in an unoriented ring of size n requires constant memory and it converges in $O(n)$ time steps and with $O(kn)$ total moves.*

Strategy for M. We now very briefly illustrate the best strategy for the malicious agent M, in order to delay as much as possible the gathering of the honest agents. Thus, we will be considering time constraints. If $k-1$ agents have the same CW orientation, M moves CW up to when it is blocked by an agent and stops. If all k agents have the same orientation, M moves in that direction up to when it is blocked by an agent, then it waits until the agent moves (as it has become a *FOLLOWER* of an *R_MSG*), and M follows the agent until it is blocked again and stops. This happens after a *HEAD* has crossed twice the node $\otimes$ and has become *STOPPED*. Finally, if agents are split in two groups, and k is odd, M moves in the direction of the group of even size; if k is even M alternatively blocks *HEAD* agents from one or the other side, so that both *HEAD* agents start a round as soon as possible with the effect of maximizing the number of rounds. Let us call this algorithm *BlockAgents* for M.

Lemma 2. *Algorithm BlockAgents provides the best strategy for the malicious agent to delay as much as possible the gathering of the honest agents.*

Results for the oriented rings, and for the case $k = 2$ in unoriented rings. For lack of space, we just very shortly illustrate two new cases, more details may be found in [12]. Let us first consider the case of the oriented ring. First note that there are no unsolvable cases, given that the symmetric configurations of Lemma 1 never arise when agents move in the same direction. One solution for $k \geq 2$ would be to apply Algorithm 1 of [5] which might require that the agents reverse direction three times when they are blocked by M. Moreover agents will terminate but do not know when there is global termination. Another solution for $k \geq 3$ would be to apply Algorithm *CollectAgents* in the case in which all the agents move in the same direction. This solution could even be improved: After an agent has become *HEAD* it just stops and waits, since if all the agents move in the same direction, then *HEAD* will not need to surround M. We have devised another simpler algorithm that requires less rounds, assures global termination, works for $k \geq 2$ agents, and uses only constant memory. The general idea is as follows: Suppose all the agents start moving in the CW direction, then the first agent that is blocked by M, let us call it A, becomes the leader and starts moving in the CCW direction. Then, when A meets an agent that is moving in the CW direction, let us call it B, B stops, while A continues to move until it is blocked again by M. Then, A reverses direction and moves back. The rendezvous

is achieved at the node of B. Note that, also in this case, an agent that arrives in $\otimes$ stops to block M. Finally, to assure that when A moves in the CCW direction it meets any other agent, e.g. B, that is moving in the CW direction, we assume that A waits at the even clock ticks and moves at the odd clock ticks, while B moves at the even clock ticks and wait at the odd clock ticks.

Let us now consider the case $k = 2$ in the unoriented ring. Note that, in the unoriented ring with asynchronous agents the problem is unsolvable for any even number $k \geq 2$, of honest agents, even if the agents know k [5]. With synchronous agents we have the following. For n odd Lemma 1 proves that the problem is not solvable, for n even we now briefly discuss a solution that requires $O(\log n)$ bits of memory, and assumes that the agents know that $k = 2$. The algorithm works as follows. It uses the general idea of the *Controlled Distance* algorithm in a ring [14]. The agents move back and forth for a certain distance x which increases at each round, in particular we choose $x = 2^i$, for $i = 0, 1, 2, \ldots$. At each round i both agents, that we call A and B, try to move for x steps in the CW direction, i.e., for x cycles of clock either they move, or they wait if they are blocked by M. Then, they move back in the CCW direction for x steps, or they wait if they are blocked by M in the opposite direction. During this walk if they meet each other they gather. Note that, if they have waited some time moving CW, then they might go back passing the starting position. It is possible to prove that eventually the two agents will either meet at some intermediate node, or they will eventually surround M and, since they are synchronized, and they start moving in CCW direction at the same time step, they will meet at the middle node, opposite to M in the ring. To prevent agents to infinitely increase value x when they start in the same direction, an agent stops when it reaches $\otimes$ twice so that the other agent will meet it at the node $\otimes$.

4 Implementation

In this section we illustrate a proof-of-concept implementation based on the Lego Mindstorms EV3 platform,[4] which is widely adopted in robotic courses.

We have represented each undirected edge of the ring as two parallel links, so that robots moving in two opposite directions on the same edge do not collide. The nodes are all identical, and they have been represented as circles that can be traversed by the agents moving around the border. The special node is labeled with a yellow marker. The honest robots are all identical, whereas the malicious agent has been physically modified to be easily noticeable.

To built our robots we have used standard Lego EV3 components. We have used the *EV3 Color Sensor* to detect the lines representing the edges of the graph so to let the robots move along them. We have used the *EV3 Infrared Seeking Sensor* and *EV3 Ultrasonic Sensor* to detect the other honest agents in the same node, and to avoid collisions when robots are following each other. By

[4] Lego mindstorms ev3, 2016. `http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3`

rotating the sensors on top of the *EV3 Servo Motors* we are also able to detect robots placed crosswise.

Face to face communication is implemented using the built-in Bluetooth adapters. However, given the small physical dimension of the ring we could not use the Received Signal Strength Indicator (RSSI) of the Bluetooth devices to connect to the robots in the same node, since all devices were showing very similar signal strengths. We have then used a centralized server that mediates connections and only enables face to face communication in the same node. We claim that on a bigger network the strength indicator would work more reliably and local communication might be implemented without resorting to a centralized server.

The software platform used for the implementation is *ev3dev*,[5] an open-source Linux-based operating system fully compatible with Lego Mindstorm robots. EV3 hardware can be controlled from a wide range of common programming languages providing bindings for the ev3dev device API. We decided to implement our algorithms in Python3 since it offers high code readability and enables rapid prototyping of applications. The source code written to evaluate the protocols consists of around 600 lines and is publicly available at our github repository page,[6] together with some exemplifying videos.[7]

5 Conclusions

In this paper we have presented the problem of gathering a set of mobile agents in presence of mobile transient faults, represented by a mobile malicious agent. We have studied the problem in oriented and unoriented ring networks, and we have shown that the proposed solutions are realistic by giving a proof-of-concept implementation of the algorithms with real Lego Mindstorms EV3 robots. We are presently studying the problem of gathering in an unoriented ring with $k = 2$ agents and constant memory, and the gathering of synchronous agents in other topologies.

References

1. S. Alpern and S. Gal. Searching for an agent who may or may not want to be found. *Operations Research*, 50(2):311–323, 2002.
2. J. Chalopin, Y. Dieudonne, A. Labourel, and A. Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29:187–205, 2016.
3. R. Cohen and D. Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. *SIAM J. on Computing*, 38:276–302, 2008.
4. J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. In *Proc. of 21st Annual ACM-SIAM Symp. on Discrete Algorithms*, 2010.

[5] Lego EV3 compatible operating system, `http://www.ev3dev.org/`

[6] Source code `https://github.com/secgroup/MTFGatheRing`

[7] Video of the rendezvous in an unoriented ring, `https://github.com/secgroup/MTFGatheRing/raw/master/videos/robots_unoriented.mp4`

5. S. Das, F.L. Luccio, and E. Markou. Mobile agents rendezvous in spite of a malicious agent. In *11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (Algosensors 2015)*, LNCS 9536, pages 211–224. Springer, Patras, Greece 2015.
6. S. Das, M. Mihalak, R. Sramek, E. Vicari, and P. Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *OPODIS*, LNCS 5401, pages 463–480, 2008.
7. Y. Dieudonne, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.
8. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, 2007.
9. P. Flocchini and N. Santoro. Distributed security algorithms for mobile agents. In Jiannong Cao and Sajal K. Das, editors, *Mobile Agents in Networking and Distributed Computing*, chapter 3, pages 41–70. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2012.
10. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. *TCS*, 384(2-3):201–221, 2007.
11. F. L. Luccio. Contiguous search problem in sierpinski graphs. *Theory of Comp. Sys.*, (44):186–204, 2009.
12. Davide Moro. From theory to practice: Solving the rendezvous problem using real robots. Master's thesis, DAIS, Università Ca' Foscari, Venezia, June 2016.
13. A. Pásztor. Gathering simulation of real robot swarm. *Tehnicki vjesnik*, 21(5):1073–1080, 2014.
14. Nicola Santoro. *Design and analysis of distributed algorithms*, volume 56. John Wiley & Sons, 2006.

Improved Protocols for Luminous Asynchronous Robots*

Mattia D'Emidio[1], Gabriele Di Stefano[2], Daniele Frigioni[2], Alfredo Navarra[3]

[1] Gran Sasso Science Institute (GSSI)
Viale Francesco Crispi 7, I–67100, L'Aquila, Italy.
mattia.demidio@gssi.infn.it
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
Università degli Studi dell'Aquila, Via Vetoio, I–67100 L'Aquila, Italy.
{gabriele.distefano,daniele.frigioni}@univaq.it
[3] Dipartimento di Matematica e Informatica, University of Perugia,
Via Vanvitelli 1, I–06123, Perugia, Italy.
alfredo.navarra@unipg.it

Abstract. The distributed setting of computational mobile entities, called robots, that have to perform tasks without global coordination has been extensively studied in the literature. A well-known scenario is that in which robots operate in *Look-Compute-Move* (LCM) cycles. LCM cycles might be subject to different temporal constraints dictated by the considered schedule. The classic models for the activation and synchronization of mobile robots are the well-known *fully-synchronous*, *semi-synchronous*, and *asynchronous* models.
In this paper, we concentrate on the weakest asynchronous model, and propose improved and general protocols to solve tasks when the robots are endowed with lights, i.e. they are *luminous*.

1 Introduction

The distributed setting of computational mobile robots that have to perform tasks without global coordination has been extensively studied in the literature. A well-known scenario is that in which robots operate in *Look-Compute-Move* (LCM) cycles (see [1,2,11,12] and references therein). During each cycle, a robot acquires

* The work has been partially supported by the European project "Geospatial based Environment for Optimisation Systems Addressing Fire Emergencies" (GEO-SAFE), contract no. H2020-691161, and by the Italian project "RISE: un nuovo framework distribuito per data collection, monitoraggio e comunicazioni in contesti di emergency response", Fondazione Cassa Risparmio Perugia, code 2016.0104.021.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 136–148 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

a snapshot of the surrounding environment (*Look* phase), then executes an appropriate algorithm by using the obtained snapshot as input (*Compute* phase), and finally moves toward a desired destination, if any (*Move* phase). Look-Compute-Move cycles might be subject to different temporal constraints dictated by the considered schedule. The classic models for the activation and synchronization of mobile robots are the well-known *fully-synchronous* (FSYNC), *semi-synchronous* (SSYNC), and *asynchronous* (ASYNC) models (see, e.g., [3,5,8]).

- **Fully-synchronous** (FSYNC): The *activation* phase (i.e. the execution of an LCM cycle) of all robots can be logically divided into global rounds. In each round all the robots are activated, obtain the same snapshot of the environment, compute and perform their move. Notice that, this assumption is computationally equivalent to a fully synchronized system in which robots are activated simultaneously and all operations happen instantaneously.
- **Semi-synchronous** (SSYNC): It coincides with the FSYNC model, with the only difference that not all robots are necessarily activated in each round.
- **Asynchronous** (ASYNC): The robots are activated independently, and the duration of each *Compute*, *Move* and inactivity phase is finite but unpredictable. As a result, robots do not have a common notion of time. Moreover, they can be seen while moving, and computations can be made based on obsolete information about positions.

Recently, a new model has been introduced by Das et al. in [4], extending the classic ones. In detail, given a model $\mathcal{M} \in \{\text{FSYNC}, \text{SSYNC}, \text{ASYNC}\}$, the authors define model $\mathcal{M}^c$, where each robot operating in $\mathcal{M}$ is equipped with a *light* that is visible to itself and to the other robots during the *Look* phase. The light associated with a robot can generate c different colors (for some constant integer $c > 0$), and can be updated by a robot during its *Compute* phase. The light is assumed to be *persistent*, i.e. despite robots can be oblivious, their lights are not automatically reset at the end of a LCM-cycle. Light-enhanced robots, introduced for the first time in [9,13], are usually referred as *luminous* robots (see, e.g., [10]). Note that, depending on the considered scenario, a robot might have visibility of the lights of either all other robots or just of a subset of them.

A first comprehensive evaluation of the computational power of robots operating in the LCM model and moving within the *Euclidean plane*, under different levels of synchronization, has been proposed in [4]. In detail, the authors provide a series of results that prove relations between classic models and variations of them, including the possibility that robots are *luminous*.

In [6] a characterization of the computational power of robots moving on *graphs* has been proposed. In particular, the authors first show relations among the three classic activation and synchronization models; second, they compare the models where robots are endowed with lights against the models without lights; third, they highlight the relations among the different models concerning luminous robots; finally, they provide a detailed comparison of the proposed results with the case of robots moving in the Euclidean plane.

1.1 Our Contribution

In this paper, we extend the work done in [6] and [7] as follows. First, we propose a reviewed and improved version of the proof given in [7] to show that FSYNC is not more powerful than $\text{ASYNC}^{O(1)}$. To this aim, we introduce a new algorithm which uses less colors to solve the same problem considered in the proof, thus showing it is solvable in ASYNC^3 rather than in ASYNC^4. Second, we generalize the newly introduced algorithm into a general algorithmic framework that for any $k > 2$ allows (any number of) robots operating in ASYNC^k to address all tasks within class of *basic formation problems* having k states (BFP_k from now on). In such problems, described for the first time in [6], the focus is on allowing robots to achieve specific sequences of k placements regardless of the movements they perform to reach each disposal. Robots might move on graphs or on the Euclidean plane. Finally, we show, by means of a specific distributed task, that there is a non-trivial subset of problems in BFP_k for which using robots in ASYNC^3 instead of ASYNC^k suffice to reach the corresponding goal.

1.2 Structure of the Paper

The paper is organized as follows. In Section 2, we provide the necessary notation for the considered problems. In Section 3, we give the improved version of the algorithm given in [7], while in Section 4 we introduce its generalization. In Section 5, we discuss on how ASYNC^3 robots can be used to solve all tasks within a non-trivial subset of BFP_k. Finally, Section 6 concludes the paper.

2 Preliminaries

We consider a system composed of mobile entities, called *robots*, that operate in LCM-cycles. In particular, each robot is modeled as an independent computational unit, capable of performing local computations. The robots are placed in a spatial environment which is assumed to be either the Euclidean plane or an undirected graph $G = (V, E)$, i.e. robots are placed on the nodes of the graph. Each robot has its own local perception of the surrounding environment, which means it can detect all other robots either as points in the plane with respect to its own coordinate system or perceiving a graph isomorphic to G and understand whether a node is occupied by a robot or not. Each robot is equipped with sensing capabilities that return a *snapshot* of the relative positions of all other robots with respect to its location.

In the remaining of the paper, we assume that robots are *anonymous* and *identical*, i.e. they are indistinguishable by their appearance, and execute the same algorithm. Unless differently specified, robots are assumed to be *oblivious*, i.e. they have no memory. Moreover, we consider robots acting without a central control, i.e. they are assumed to be *autonomous* and not able to directly communicate information (e.g. by a wireless interface) with other robots, i.e. they are *silent*. Each robot is endowed with motor capabilities and can freely move. However,

when moving on graphs, the movement along one edge is considered instantaneous, so that each time a robot perceives the snapshot of the current configuration, it sees all other robots always on the nodes of the graph. We will specify different assumptions if required by the context.

At any point in time, a robot is either *active* or *inactive*. All robots are initially inactive, i.e. they are idle. When active, a robot executes an LCM-cycle by performing the following three operations in sequence, each of them associated with a different state:

- **Look:** The robot observes the environment. The result of this phase is a snapshot of the positions of all robots with respect to its own perception.
- **Compute:** The robot executes its own algorithm, using the data sensed in the *Look* phase as input. The result of this phase is a target node among the neighbors of the node in which the robot currently resides (at most one edge per cycle can be traversed by a robot).
- **Move:** The robot moves toward the computed target. If the target is the current position, then the robot stays still, i.e. it performs what is called a *null* movement.

The amount of time to complete a full LCM-cycle is assumed to be finite but unpredictable.

3 Algorithm 3-FORTHBACK

In this section, we propose a reviewed and improved version of the proof given in [7] to show that FSYNC is not more powerful than $\text{ASYNC}^{O(1)}$. To this aim, we make use of the following task for the proof.

Forth and Back (FB)	
***Input*:**	Two anonymous robots placed at two distinct internal nodes of a path P at some distance d (in terms of number of edges).
***Solution*:**	A distributed algorithm that ensures the two robots to alternate their distance between d and $d+2$.

In [6] it has been already shown that the FB problem cannot be solved in FSYNC. They also show that FB can be solved in $\text{ASYNC}^{O(1)}$, by an algorithm (namely FORTHBACK) which requires each robot to be equipped with a light that can assume *four* colors. Here, we propose an enhanced algorithm, named Algorithm 3-FORTHBACK (see Algorithm 1), which is able to solve the problem even if the robots are endowed with a light that can generate only *three* colors.

The new strategy has its own practical interest since it improves over the algorithm given in [6]. In particular, it reduces the number of colors needed for solving the problem, which can be easily imagined as a proxy for the use of power/communication resources. In addition, the intuition behind its design opens new perspectives for devising general strategies dedicated to robots operating in ASYNC^{k}. We will discuss in the next sections on such aspect. In details,

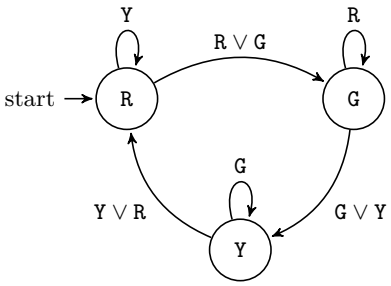


Fig. 1. Finite state machine associated to algorithm 3-FORTHBACK.

we will show that the technique underlying Algorithm 3-FORTHBACK can be generalized into an algorithmic framework that allows ASYNC^k robots to address a class of problems of significant interest.

We now proceed with the description of the new algorithm. The three colors used by Algorithm 3-FORTHBACK are RED (R), GREEN (G), and YELLOW (Y), with the following meanings:

- RED indicates that the robot is ready to move for the next step and the previous distance must be increased;
- GREEN indicates that the robot is ready to move for the next step and the previous distance must be decreased;
- YELLOW indicates that the robot has moved to decrease the previous distance and it is ready for the next step.

In what follows, we denote by $L[r]$ the light associated with a given robot r. At the beginning, both robots start with the lights set to RED. As we will see, Algorithm 3-FORTHBACK ensures that, whenever one of the robots has the light set to RED (to GREEN, respectively), the other robot will eventually turn its light to RED (to GREEN, respectively), i.e. they are always able to be synchronized at some point. In other words, Algorithm 3-FORTHBACK solves FB in ASYNC^3 by exploiting the parity encoding of the current step by means of the light. We describe the algorithm as it is executed by a generic given robot r. For the sake of clarity, we also summarize the behavior of each robot via a finite state machine (see Figure 1), where the label within a node represents the color of the light $L[r]$ of the executing robot, while the label above an edge represents the condition on the light $L[r']$ of the other robot r' that triggers the transition to occur.

Theorem 1. *Algorithm* 3-FORTHBACK *correctly solves the* FB *problem in* ASYNC^3 .

Proof. First of all, notice that, if d is the initial distance between the two robots (when both lights are RED), then $d + 2$ defines the final placement of the two robots after the first step. Therefore, in order to reach the requested configuration,

Algorithm 1: Algorithm 3-FORTHBACK performed by a generic robot r to solve FB in ASYNC^3.

```
1  Let r' be the other robot;
2  Let δ be my distance from r';
3  if L[r] = RED then
4  |  if L[r'] = RED ∨ L[r'] = GREEN then
5  |  |  L[r] := GREEN;
6  |  |  Let v be the neighbor at distance δ + 1 from r';
7  |  |  The new position is v;
8  |  |  Exit;
9  if L[r] = GREEN then
10 |  if L[r'] = GREEN ∨ L[r'] = YELLOW then
11 |  |  L[r] := YELLOW;
12 |  |  Let v be the neighbor at distance δ − 1 from r';
13 |  |  The new position is v;
14 |  |  Exit;
15 if L[r] = YELLOW then
16 |  if L[r'] = YELLOW ∨ L[r'] = RED then
17 |  |  L[r] := RED;
18 |  |  Exit;
```

when a robot has to move to increase its distance (i.e. its light is RED), if the current distance is d', then the target distance has to be set to $d'+1$ (see Line 7), since both robots contribute of one edge.

Similarly, if $d+2$ is the distance between the two robots after an increasing step, then d defines the placement of the two robots after the current step. Hence, in order to reach the requested configuration, when a robot has to move to decrease its distance (i.e. its light is GREEN), if the current distance is d', then the target distance has to be set to $d'-1$ (see Line 13), since both robots contribute of one edge.

Now, if $L[r]$ is either RED or GREEN, then r is ready to accomplish a movement, which must either increase the distance of the previous placement ($L[r] =$ RED case) or decrease it ($L[r] =$ GREEN case). The robot r can decide which is the case by looking at the light of the other robot.

On the one hand, if $L[r] =$ RED and $L[r']$ is either RED or GREEN (see Line 4), then r must move away from r' of one edge, as robot r' is either ready to move to increase the distance ($L[r'] =$ RED) or is ready to move to decrease the distance($L[r'] =$ GREEN). On the other hand, if $L[r] =$ GREEN and $L[r']$ is either GREEN or YELLOW (see Line 10), then r must move closer to r' of one edge since robot r' is either ready to move to decrease the distance ($L[r'] =$ GREEN) or has already accomplished a movement that has decreased it ($L[r'] =$ YELLOW).

If $L[r]$ is YELLOW and $L[r']$ is either RED or YELLOW, then r can conclude that r' is either ready to move to increase the distance ($L[r'] =$ RED) or has already terminated a movement whose purpose was that of decreasing the distance

($L[r'] = \textsc{Yellow}$). Robot r, in this case, just switch its light to RED (see Line 17). If r' has performed the above step before r, i.e. $L[r] = \textsc{Yellow}$ and $L[r']$ is GREEN, then, r simply keeps $L[r]$ to YELLOW. □

4 Generalizing Algorithm 3-FORTHBACK

In this section, we propose a general algorithmic paradigm that allows robots operating in ASYNC^k, $k > 2$ to address a whole class of problems, namely the basic formation problems BFP_k.

BFP_k problems can be informally defined as the class of problems where a set of k static configurations have to be (possibly cyclically) reached, in order to achieve the goal, regardless of the movements they perform to reach each disposal (see [6] for a more thorough discussion). We remind that the class of BFP_k problems can be defined for robots moving on both graphs and the Euclidean plane.

More formally, problems are in BFP_k if and only if their dynamics can be completely described by a finite state machine (o by any subset of it) with the following characteristics:

- there are k distinct states;
- there exists a total (strict) ordering among the k states, i.e. for every i–th state there exists a transition that brings the system from state i to a state $(i+1) \mod k$.

In other words, the problem asks the robots to change from a state to another one in a sequential manner, according to some criteria. An example of finite state machine of the above kind is reported in Figure 2, where label x_i within a node represents the i–the state in the ordering. Clearly the above class of problems satisfies $\mathrm{BFP}_1 \subseteq \mathrm{BFP}_2 \subseteq \cdots \subseteq \mathrm{BFP}_k$ for any k.

Trivially, the FB problem, discussed in Section 3, belongs to BFP_2, since the robots have to cyclically oscillate between $k = 2$ distinct configurations, i.e. those in which the robots are at distance $d' + 1$ and $d' - 1$. However, to solve FB we required 3 colors in order to synchronize the robots. Whereas, when $k > 2$ there is no need for an extra color.

Note that, problems in BFP_k might be solvable in FSYNC or not. This possibility depends on the capability of the robots of distinguishing, by only observing the surrounding environment, two states that are adjacent in the sequence (and therefore are connected by a transition in the finite state machine). If robots are capable of doing so, it is easy to see that (any number of) FSYNC robots can solve problems of BFP_k in $\Theta(k)$ time steps by the following very simple strategy. Starting from an initial state x_0, at each synchronous time step, all robots wake up and perceive the very same snapshot of the surrounding environment. Then, given the state they perceived, say x_i, they all check the criterion associated to the arcs outgoing x_i in the finite state machine (i.e. they perform the compute phase) and, according to the result they perform the move phase that either brings them into $x_{(i+1) \mod k}$ or keeps them into x_i. An example

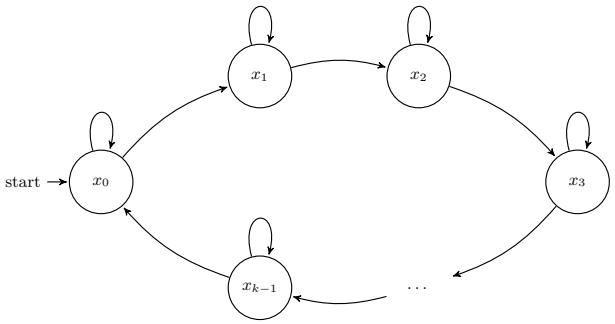


Fig. 2. Finite state machine associated to problems in BFP_k.

of problem in BFP_k that is not solvable in FSYNC is indeed problem FB, since the knowledge obtained by the acquired snapshot is not enough to determine whether the target distance has to be set to $d'+1$ or to $d'-1$ and hence to decide the next state. On the contrary, an example of problem in BFP_k that is solvable in FSYNC, by the above strategy, is the so-called *Pattern Series Chasing* (PSC) problem whose first description can be found in [7], and is reported below.

Pattern Series Chasing (PSC)

***Input*:** An undirected and complete graph G with nodes labeled from 1 to n. An array A of c patterns, for some integer constant $c > 0$, each involving $k < n$ nodes of G, such that $A[i] \neq A[j]$, for every $0 \leq i \neq j < c$. A set of k robots forming $A[0]$ in G.

***Solution*:** A distributed algorithm that ensures robots to form pattern $A[(i+1) \bmod c]$ after $A[i \mod c]$, $i \in \mathbb{N}$.

Regarding FB, in the previous section we have shown that it can be solved by two ASYNC^3 robots by Algorithm 1. By following the intuition underlying such algorithm, we now give a generalization of it that allows (any number of) ASYNC^k robots to solve problems in BFP_k for $k > 2$. The main idea behind the algorithmic paradigm we are proposing is based on the fact that, when considering BFP_k problems, an implicit ordering can be provided by the lights of ASYNC^k robots.

The strategy for solving any BFP_k by ASYNC^k robots can be summarized as follows (see Algorithm 2 for more details). Suppose we are given a generic problem in BFP_k and a set $\mathcal{R} = r_1, r_2, \ldots, r_n$ of n robots. We denote by $L[r_x]$ the color assumed by the light of robot r_x. We equip the robots with lights assuming k colors with the following meaning: color COL_i indicates that the robot is in state x_i and it is ready to move to state $x_{(i+1) \bmod k}$, and to accomplish the associated (possibly null) move phase. Moreover, we assume that colors exhibit a total (strict) ordering, i.e. for each color x_i there exist two colors $x_{(i\pm1) \bmod k}$ such that $x_{(i-1) \bmod k} < x_i < x_{(i+1) \bmod k}$. To solve the problem is then enough

Algorithm 2: Algorithmic paradigm that allows (any number of) ASYNC^k robots to solve problems in BFP_k, $k > 2$.

```
1 Let r_i be the robot executing the algorithm;
2 if L[r_i] = COL_{x_i} then
3 |   foreach r_j ∈ R : i ≠ j do
4 |   |   if L[r_j] < L[r_i] then
5 |   |   |   Exit;
6 |   L[r_i] := COL_{x_{(i+1) mod k}};
7 |   Perform move phase of state x_{(i+1) mod k};
8 |   Exit;
```

to exploit the ordering of the colors to take coherent decisions about the state to be reached. In particular, each robot r_i performs its (possibly null) move phase if and only if its current color $L[r_i] = \mathrm{COL}_{x_i}$ is less or equal than the colors of the lights of all other robots. If so, it also switches its light to $\mathrm{COL}_{x_{(i+1) \bmod k}}$. By the above discussion, the following result can be stated.

Theorem 2. ASYNC^k *robots can solve any problem in* BFP_k, $k > 2$.

It is worth noticing that Algorithm 3-FORTHBACK, given in the Section 3, can be easily derived by Algorithm 2 by considering the different move phases associated to each of the two states of problem FB.

As a final remark, note that an algorithm for solving problem PSC by ASYNC^c robots, where c is the cardinality of the array A, can be derived as well by Algorithm 2 by customizing each move phase according to the pattern to be reached.

5 Further Applications of Algorithm 3-FORTHBACK

In this section, we show that there is a non-trivial category of problems in BFP_k for which something better than using ASYNC^k robots can be done. To this aim, we first define the main characteristics of such category and show they can be solved by weaker ASYNC^3 robots. Consider again Algorithm 3-FORTHBACK defined in Section 3, the three colors have been though to encode the static configurations, but they can be used to simply encode an *advancing* in the current computational process. This is not the case for the PSC problem as shown in [7]. In order to better understand the intuition, we now consider a variant of the classical *patrolling problem*, where three robots must infinitely traverse a circle but ensuring some specific configurations where they are all idle. It means that during the patrolling, robots must ensure some predefined *static* configurations.

Given a circle C, let $arc(x, y)$ be the smallest arc of C from x to y.

Patrolling With Stops (PWS)	
***Input*:**	Three anonymous robots r_1, r_2 and r_3 placed on a circle C with center c, such that $arc(r_1, r_2) = d$, $arc(r_2, r_3) = 2d$, $d < \frac{\pi}{6}$.
***Solution*:**	A distributed algorithm that ensures the three robots to patrol C by forming a static configuration similar to C each time the angle α in c defined by the initial position of r_1, c and the current position of r_1 is $p \cdot \frac{\pi}{3}$, for any integer p.

In PWS, we have considered six static configurations that must be reached cyclically, by setting the angle α as a multiple of $\frac{\pi}{3}$. Clearly an arbitrary number of configurations can be considered by reducing α. According to Theorem 2, we know how to solve PWS in ASYNC^6. We now show that an algorithm similar to Algorithm 3-FORTHBACK can be defined to solve the PWS problem in ASYNC^3.

Theorem 3. *There exists an algorithm in* ASYNC^3 *that solves the* PWS *problem.*

Proof. The proof proceeds by providing three subroutines of the same algorithm, each one executed by a different robot, according to its role among r_1, r_2 and r_3. Although the three robots are anonymous, by looking to their relative positioning and lights, they can always deduce who they are. As we are going to show, our algorithm never changes the role of a robot. The meaning of the lights is the same for all robots, that is:

- RED: ready to reach the new static configuration
- GREEN: moving to the computed target
- YELLOW: target reached

The minimum arc of C containing all three robots is always divided into two sub-arcs. The middle robot is always r_2. Initially (when all robots assume light RED), the closest robot to r_2 is r_1, while the other is r_3. The proposed algorithm makes r_3 move always as first, increasing its distance from r_2. This is done by switching $L[r_3]$ to GREEN and evaluating the next position where a static configuration must be guaranteed. Movements are always performed along the circumference of C as the patrolling requires.

While $L[r_3] =$ GREEN, r_1 and r_2 do not move. Once $L[r_3] =$ YELLOW, r_2 can start its movement toward the next position. Due to the adversary, each time a robot moves toward a target position, it can reach it or it can be stopped before. Nevertheless, our algorithm is designed so as the same robot will be the unique one allowed to move until it reaches the desired destination, eventually.

Once both r_2 and r_3 have reached their current destinations and $L[r_2] = L[r_3] =$ YELLOW, the last robot that has to move deduces it is r_1.

Once all robots have reached their destinations, $L[r_1] = L[r_2] = L[r_3] =$ YELLOW and the configuration is static, that is a stop has been performed and the next positioning can start. This is done first by switching all the lights to RED, but in a sequential order, starting from r_3, then r_2 and finally r_1. Then, the whole process is repeated. □

Algorithm 3: Algorithm PWS $\{r_1\}$ performed by r_1

Let x be the point on C between r_1 and r_2 such that $|arc(x, r_3)| = \frac{3}{2}|arc(r_2, r_3)|$;
if $(L[r] = \text{RED} \wedge L[r_2] = L[r_3] = \text{YELLOW}) \vee (L[r] = \text{GREEN} \wedge |arc(r, r_3)| > x)$ **then**
    $L[r] := \text{GREEN}$;
    The new position is x;
    Exit;
if $L[r] = \text{GREEN}$ **then**
    $L[r] := \text{YELLOW}$;
    Exit;
if $L[r] = \text{YELLOW} \wedge L[r_2] = L[r_3] = \text{RED}$ **then**
    $L[r] := \text{RED}$;
    Exit;

Algorithm 4: Algorithm PWS $\{r_2\}$ performed by r_3

Let x be the point on C between r_2 and r_3 such that $|arc(x, r_3)| = \frac{2}{3}\left(|arc(r_1, r_3)| - \arcsin\frac{\pi}{3}\right)$;
if $(L[r] = \text{RED} \wedge L[r_3] = \text{YELLOW}) \vee (L[r] = \text{GREEN} \wedge |arc(r, r_3)| > x)$ **then**
    $L[r] := \text{GREEN}$;
    The new position is x;
    Exit;
if $L[r] = \text{GREEN}$ **then**
    $L[r] := \text{YELLOW}$;
    Exit;
if $L[r] = L[r_1] = \text{YELLOW} \wedge = L[r_3] = \text{RED}$ **then**
    $L[r] := \text{RED}$;
    Exit;

6 Conclusion

In this paper, we have considered the problem of devising protocols for luminous asynchronous robots. In details, we have extended the work done in [6] and [7] in three directions. We first have proposed a reviewed, improved version of the proof given in [7] to show that FSYNC is not more powerful than $\text{ASYNC}^{O(1)}$. To do so, we have introduced a new more efficient algorithm for solving the FB problem that requires less colors to work with respect to to its previous counterpart of [7]. Apart from the desirable property of being optimized in terms of colors, the new algorithm has driven us also to the design of its generalization that allows (any number of) ASYNC^k robots to address any problem in BFP_k, for any $k > 2$. As a final contribution, we have shown that there exists a non-trivial subset of problems in BFP_k for which weaker ASYNC^3 robots are powerful enough to achieve the considered goal.

Algorithm 5: Algorithm PWS $\{r_3\}$ performed by r_3

Let x be the point on C such that $|arc(r_1, x)| = 3|arc(r_1, r_2)| + \arcsin \frac{\pi}{3}$;
if $(L[r] = \text{GREEN} \wedge |arc(r_1, r)| < |arc(r_1, x)|) \vee L[r] = L[r_1] = L[r_2] = \text{RED}$ **then**
    $L[r] := \text{GREEN}$;
    The new position is x;
    Exit;
if $L[r] = \text{GREEN}$ **then**
    $L[r] := \text{YELLOW}$;
    Exit;
if $L[r] = L[r_1] = L[r_2] = \text{YELLOW}$ **then**
    $L[r] := \text{RED}$;
    Exit;

References

1. A. Chatterjee, S. G. Chaudhuri, and K. Mukhopadhyaya. Gathering asynchronous swarm robots under nonuniform limited visibility. In *11th International Conference on Distributed Computing and Internet Technology (ICDCIT)*, volume 8956 of *Lecture Notes in Computer Science*, pages 174–180. Springer, 2015.
2. S. Cicerone, G. Di Stefano, and A. Navarra. Minmax-distance gathering on given meeting points. In *9th Int. Conference on Algorithms and Complexity (CIAC)*, volume 9079 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2015.
3. G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering on rings under the look-compute-move model. *Distributed Computing*, 27(4):255–285, 2014.
4. S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609:171–184, 2016.
5. B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer. A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In *23rd ACM Symp. on Parallelism in algorithms and architectures (SPAA)*, pages 139–148. ACM, 2011.
6. M. D'Emidio, D. Frigioni, and A. Navarra. Characterizing the computational power of anonymous mobile robots. In *36th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 293–302, 2016.
7. M. D'Emidio, D. Frigioni, and A. Navarra. Synchronous robots vs asynchronous lights-enhanced robots on graphs. In *16th Italian Conference on Theoretical Computer Science (ICTCS)*, volume 322 of *Electronic Notes in Theoretical Computer Science*, pages 169–180. Elsevier, 2016. doi:10.1016/j.entcs.2016.03.012.
8. S. Devismes, F. Petit, and S. Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. In *16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 5869 of *Lecture Notes in Computer Science*, pages 195–208, 2009.
9. A. Efrima and D. Peleg. Distributed models and algorithms for mobile robot systems. In *33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 4362 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2007.

10. P. Flocchini. Computations by luminous robots. In *14th International Conference on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW)*, volume 9143 of *Lecture Notes in Computer Science*, pages 238–252. Springer, 2015.
11. P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3(2):1–185, 2012.
12. S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Gathering an even number of robots in an odd ring without global multiplicity detection. In *Mathematical Foundations of Computer Science (MFCS)*, pages 542–553. Springer, 2012.
13. D. Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In *7th International Workshop on Distributed Computing (IWDC)*, volume 3741 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

Active Spreading in Networks

G. Cordasco[2], L. Gargano[1], and A. A. Rescigno[1]

[1] Department of Informatics, University of Salerno, Italy,
[2] Department of Psychology, Second University of Naples, Italy.

Abstract. Identifying the most influential spreaders is an important issue for the study of the dynamics of information diffusion in complex networks. In this paper we analyze the following spreading model. Initially, a few nodes know a piece of information and are *active* spreaders of it. At subsequent rounds, spreaders communicate the information to their neighbors. Upon receiving the information, a node becomes *aware* of it but does not necessarily become a spreader; it starts spreading only if it gets the information from a sufficiently large number of its neighbors. We study the problem of choosing the smallest set of initial spreaders that guarantee that all the nodes become aware of the information. We provide hardness results and show that the problem becomes tractable on trees. In case of general graphs, we provide an efficient algorithm and validate its effectiveness (in terms of the solution size) on real-life networks.

1 Introduction

During the past decade spreading processes in complex networks have experienced a particular surge of interest. A large part of research activity in the area deals with the analysis of influence spreading in social networks. There are many situations where members of a network may influence their neighbors' behavior and decisions, by swaying their opinions, by suggesting what products to buy, or simply by passing on a misinformation [7,23,30]. A key research question, related to understand and control the spreading dynamics, is how to efficiently identify a set of users that can diffuse information within the network. This is the problem addressed in this paper. Our scenario posits a population consisting of n individuals that, with respect to the information, are subdivided into *ignorant, aware*, and *spreading*. Initially, all individuals are ignorant. Then an initial set of spreaders is selected. When a spreader informs an ignorant node v, the ignorant node v becomes aware; as soon as the individual v is informed by a number of spreaders greater than a threshold $t(v)$, it starts spreading the information itself. The motivations that lead us to consider such a scenario come from experimental

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 149–162 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

studies of how information spread in social networks. Indeed, information doesn't flow freely in the network but it requires active sharing which, in turn, depends on individual conviction to pass it on. We refer to [2] for a study of how exposure to social signals affects diffusion.

We model the network as an undirected graph $G = (V, E)$, where V is the set of individuals and the set of edges E represents the relationships among members of the network, i.e., $(u, v) \in E$ if individuals u and v can directly communicate. We posit a threshold function $t : V \to \{0, 1, 2, \ldots\}$, and we denote by $N(v)$ the neighborhood of $v \in V$. An active diffusion process starting at $S \subseteq V$ is a sequence of node subsets: $\mathsf{Spreader}_G[S, \tau]$, $\tau = 0, 1, \ldots$, such that $\mathsf{Spreader}_G[S, 0] = S$ and

$$\mathsf{Spreader}_G[S, \tau] = \mathsf{Spreader}_G[S, \tau - 1] \cup \Big\{ u \, s.t. \, \big|N(u) \cap \mathsf{Spreader}_G[S, \tau - 1]\big| \geq t(u) \Big\},$$

for $\tau \geq 1$. The process terminates when $\mathsf{Spreader}_G[S, \rho] = \mathsf{Spreader}_G[S, \rho - 1]$ for some $\rho > 1$. We denote by $\mathsf{Spreader}_G[S] = \mathsf{Spreader}_G[S, \rho]$. Hence, when the process stops the set of aware nodes is

$$\mathsf{Aware}_G[S] = \mathsf{Spreader}_G[S] \cup \Big\{ u \, s.t. \, N(u) \cap \mathsf{Spreader}_G[S] \neq \emptyset \Big\}.$$

Given G, a threshold function $t(\cdot)$, we aim to identify a small node set $S \subseteq V$ such that $\mathsf{Aware}_G[S] = V$.[3] Namely, we consider the following problem,

Perfect Awareness (PA).
Instance: A graph $G = (V, E)$, node thresholds $t : V \longrightarrow \mathbb{N}_0$.
Question: Find a seed set $S \subseteq V$ of minimum size such that $\mathsf{Aware}[S] = V$.

We refer to the set S for which $\mathsf{Aware}[S] = V$ as a perfect seed set and to the nodes in S as *seeds*.

1.1 Related Work and Our Results

The above algorithmic problem has its roots in the area of the *spread of influence* in Social Networks. Maximizing the spread of viral information across a network naturally suggests many interesting optimization problems (see [7,17] and references quoted therein). The first authors to study spread of influence in networks from an algorithmic point of view were Kempe *et al.* [19,20,21]. Chen [6] studied the following minimization problem: given a graph G and fixed thresholds $t(v)$, for each vertex v in G, find a set of minimum size that eventually influences all (or a fixed fraction of) the nodes of G. This problem is usually referred as the Target Set Selection Problem (TSS). He proved a strong inapproximability result that makes unlikely the existence of an algorithm with approximation factor better than $O(2^{\log^{1-\epsilon}|V|})$. Chen's result stimulated a series

[3] In the rest of the paper we omit the subscript G whenever the graph is clear from the contex.

of papers [1,3,5,8,9,10,11,12,18,27,28] that isolated interesting cases in which the problem (and variants thereof) become tractable. Heuristics for the TSS problem that work for general graphs have been proposed in the literature [13,16,29].

However, the papers appeared in the scientific literature considered the basic model in which *any* node, as soon as it is influenced by its neighbors, it immediately starts spreading influence. In this paper we consider a more refined model that differentiates among spreaders and plain aware node. This model has been first considered in [14], where the Awareness Maximization Problem in which one asks for a set S, with $|S| \leq \beta$, that achieves the maximum awareness in the network has been studied.

In Section 2, we study the computational complexity of the PA problem and extend the TSS problem hardness result to the PA problem. In Section 3, we give an algorithm that outputs a perfect seed set for any input graph. Experimental evaluation of the proposed algorithm is given in Section 4; it shows that the proposed algorithm outperforms some heuristics developed for related problems. Finally, we show that our problem becomes tractable if the graph is a tree (Section 5).

We would like to remark that if the threshold $t(v)$ is equal to the node degree $d(v)$, for each $v \in V$, then a perfect target set for G is, indeed, a dominating set for G. Hence, the proposed algorithm outputs a *dominating set* for G and computational experiments suggest that it performs very well in practice.

2 Complexity

We prove the hardness of the PA problem by constructing a gap-preserving reduction from the TSS problem. We recall that the TSS problems, given $G = (V, E)$ with threshold function $t : V \to N$, asks to identify a minimum size $S \subseteq V$ such that $\mathsf{Spreader}[S] = V$. Our Theorem 1 follows from the inapproximability results for the TSS problem given in [6].

Theorem 1. *The PA problem cannot be approximated within a ratio of* $O(2^{log^{1-\epsilon}n})$*, for any* $\epsilon > 0$*, unless NP*$\subseteq$*DTIME*$(n^{polylog(n)})$*.*

Proof. We give a reduction from the Target Set Selection problem. Consider an instance of the TSS problem consisting in a graph $G = (V, E)$ with threshold function $t(\cdot)$. Let $V = \{v_1, \ldots, v_n\}$, we build a graph $G' = (V', E')$ as follows:

- Replace each $v_i \in V$ by a triangle in which the node set is $V'_i = \{v_{i,0}, v_{i,1}, v_{i,2}\}$. Formally,
  - $V' = \bigcup_{i=1}^{n} V'_i = \{v_{i,j} \mid 1 \leq i \leq n,\, 0 \leq j \leq 2\}$
  - $E' = \{(v_{i,0}, v_{\ell,0}) \,|\, 1 \leq i < \ell \leq n,\, (v_i, v_\ell) \in E\} \bigcup$ $\{(v_{i,j}, v_{i,\ell}) \,|\, i = 1, \ldots, n,\; 0 \leq j < \ell \leq 2\}$;
- the thresholds are $t'(v_{i,0}) = t(v_i)$ and $t'(v_{i,1}) = t'(v_{i,2}) = 2$, for $i = 1, \ldots, n$.

Notice that G corresponds to the subgraph of G' induced by the set $\{v_{i,0}|1 \leq i \leq n\}$. We show that there exists a target set $S \subseteq V$ for G iff there exists a perfect seed set $S' \subseteq V'$ for G' such that $|S'| = |S|$.
Assume first that $S \subseteq V$ is a target set for G. Since $\mathsf{Spreader}_G[S] = V$, then all the nodes $v_{i,0} \in V'_i$ will become spreaders in G' when the seed set is $S' = \{v_{i,0} \in V'|v_i \in S\}$. Once a node $v_{i,0}$ becomes a spreader the nodes $v_{i,1}, v_{i,2}$ are aware in the next round. Hence, S' is a perfect seed set for G', that is $\mathsf{Aware}_{G'}[S'] = V'$.
Assume now that $S' \subseteq V'$ is a perfect seed set for G'. Let $S'' = \{v_{i,0} \in V' \mid S' \cap V'_i \neq \emptyset\}$. It is easy to observe that $\mathsf{Aware}_{G'}[S''] = \mathsf{Aware}_{G'}[S'] = V'$. Let $V'_0 = \{v_{i,0} \mid 1 \leq i \leq n\}$. A node in V'_0 can influence at most 2 nodes in $V' - V'_0$—the other vertices of the triangle its belongs to. Hence, in order to influence all the nodes in $V' - V'_0$ all nodes in V'_0 must be spreaders, that is, $\mathsf{Spreader}_{G'}[S''] = V'_0$. As a consequence, recalling that G is isomorphic to the subgraph of G' induced by V'_0, we get $\mathsf{Spreader}_G[\{v_i \mid S' \cap V'_i \neq \emptyset\}] = V$. □

We notice that the Target Set Selection problem remains hard when each node has threshold upper bounded by a constant; in particular, it was proved in [6] that approximating it when each node has threshold at most 2 is as hard as approximating the problem in the general setting, even for constant degree graphs. Our reduction allows to extend this result as well, namely one has that the PA problem remains hard to approximate even if all nodes have threshold at most 2.

3 A general algorithm for the PA problem

In this section we propose an algorithm for the PA problem in case of arbitrary graphs and thresholds. The algorithm PA(G, t), given in Algorithm 1, works greedily by iteratively deprecating nodes from the input graph G unless a certain condition occurs which makes a node be added to the seed set S; it stops when all nodes have either been discarded or selected as seed.

The algorithm maintains five sets of nodes: S that represents the current seed set; U that represents the set of nodes in the surviving graph (i.e., nodes not removed from the initial graph); $Temp$ which is a set of nodes moved into a *temporary waiting* state (such nodes still belong to U but their neighbors will not count on them for being influenced); R that represents a set of nodes that must become spreaders (but will not do so with the current seed); A is the set of aware nodes (assuming that all the nodes in R will be indeed spreaders).

The algorithm proceeds as follows: As long as there exists at least a non-aware node or there is a node in R, a node v is selected according to a certain function (see Case 3) and is moved into a *temporary waiting* state, represented by the set $Temp$. As a consequence of being in $Temp$, all the neighbors of v will not count on v for being influenced (for each $u \in N(v)$ the value $\delta(u)$, which denotes the degree of u restricted to the nodes in the $U - Temp$, is reduced by 1).

Due to this update, some nodes in the surviving graph may remain with less "*usable*" neighbors (if a node $v \notin A$ has $\delta(v) = 0$ or $v \in R$ has $\delta(v) < t(v)$); in such a case (see Case 2) the nodes are added to the seed set and removed from

the graph, while the thresholds of their neighbors are decreased by 1 (since they receive v's influence).

If (see Case 1) the surviving graph contains a node v whose threshold has been decreased down to 0 (which means that the nodes which have been already added to the seed set S – see Case 2 – suffice to make v a spreader), v is deleted from the graph and the thresholds of its neighbors are decreased by 1 (since once v becomes a spreader, they will receive its influence). Notice that Case 1 can also apply to nodes in $Temp$.

Algorithm 1: PA(G, t) //$G = (V, E)$ is a graph with thresholds $t(v)$ for $v \in V$

```
1  S = ∅; Temp = ∅; U = V; R = ∅; A = ∅;
2  foreach v ∈ V do
3  |  k(v) = t(v);
4  |  δ(v) = |N(v)|;
5  while A ≠ V OR R ≠ ∅ do
6  |  if ∃v ∈ U s.t. k(v) = 0 then   // Case 1): v is a spreader, thanks to its
   |   neighbors outside U
7  |  |  foreach u ∈ N(v) ∩ U do
8  |  |  |  k(u) = max(k(u) − 1, 0); A = A ∪ {u};
9  |  |  |  if v ∉ Temp then δ(u) = δ(u) − 1;
10 |  |  U = U − {v};   R = R − {v};   A = A ∪ {v};
11 |  else
12 |  |  if ∃v ∈ (U−Temp) ∩ R s.t. δ(v)<k(v) OR ∃v ∉ A s.t. δ(v) = 0
   |  |   then
   |  |  |                                   // Case 2): v must be a seed
13 |  |  |  S = S ∪ {v};
14 |  |  |  foreach u ∈ N(v) ∩ U do
15 |  |  |  |  k(u) = k(u) − 1;
16 |  |  |  |  δ(u) = δ(u) − 1;
17 |  |  |  U = U − {v};   R = R − {v};   A = A ∪ {v};
18 |  |  else
19 |  |  |  if U − Temp − R ≠ ∅ then   // Case 3): v is moved in the
   |  |  |   temporary repository
20 |  |  |  |  v = argmin_{w∈U−Temp−R} {δ(w)}
21 |  |  |  |  if v ∉ A then
22 |  |  |  |  |  R = R ∪ {u} where u = argmax_{w∈N(v)∩(U−Temp)} {δ(w)}
23 |  |  |  |  |  foreach z ∈ N(u) ∩ U do A = A ∪ {z};
24 |  |  |  else
25 |  |  |  |  v = argmax_{w∈R} { k(w) / (δ(w)(δ(w)+1)) };
26 |  |  |  foreach u ∈ N(v) ∩ U do δ(u) = δ(u) − 1;
27 |  |  |  Temp = Temp ∪ {v};   R = R − {v};   A = A ∪ {v};
28 return S
```

In such a case the value of $\delta()$ of the neighbors of the selected node v were already reduced by 1—when v moved to $Temp$—and, therefore, it is not reduced further. By construction, once a node is moved to $Temp$, then it will be removed from the graph only by Case 1; indeed, Case 2 and 3 only apply to nodes outside $Temp$. In other words, nodes moved to $Temp$ will never belong to the seed set.

When Case 3 applies the idea is to identify nodes that will never belong to the initial seed set. Two cases are considered, if the surviving graph still contains nodes which do not belong to the set R, then one of such nodes having minimum $\delta()$ is moved to the set $Temp$. Otherwise all the nodes in the surviving graph must spread and the choice of the node to be deprecated is made according to a metric first studied in [15]. We notice that the metric used to choose which node to deprecate, that is to pose in the temporary repository when Case 3 applies, does not influence the correctness of the algorithm but it is the hearth for its effectiveness in terms of solution size.

Example 1. Let G be a complete graph, the algorithm $\mathrm{PA}(G, t)$ optimally returns a single seed: At the first iteration of the while loop, Case 3) applies and a node v_1 is selected; then a node v_2 is marked as required while all the others—being neighbors of v_2—are marked aware; during the successive iterations, $|V|-t(v_2)-1$ nodes are removed from U; finally Case 2) holds for v_2 which is added to S and the algorithm returns $S = \{v_2\}$.

In the rest of the paper, we use the following notation. We denote by n the number of nodes in G, that is, $n = |V|$ and by λ the number of iterations of the while loop of algorithm $\mathrm{PA}(G, t)$. Given a subset $V' \subseteq V$ of vertices of G, we denote by $G[V']$ the subgraph of G induced by nodes in V'. Moreover, with respect to the iterations of the while loop in $\mathrm{PA}(G, t)$, for each $i = 1, \ldots, \lambda$ we denote:

- by v_i the node selected during the i-th iteration;
- by $U_i, Temp_i, S_i, R_i, A_i, \delta_i(u)$, and $k_i(u)$, the sets $U, Temp, S, R, A$ and the values of $\delta(u), k(u)$, respectively, as updated at the beginning of the i-th iteration.

When $i = 1$, the above values are those of the input graph G, that is: $U_1 = V$, $G[U_1] = G$, $\delta_1(v) = |N(v)|$ and $k_1(v) = t(v)$, for each node v of G.

The following properties will be useful for the algorithm analysis.

Fact 1 *For each iteration i of the while loop in PA(G, t),*
1. $V - U_i \subseteq A_i$ **2.** $Temp_i \subseteq A_i$ **3.** $R_i \subseteq U_i - Temp_i$

Fact 2 *For each iteration i of the while loop in PA(G, t) and $u \in U_i$, it holds*

$$\delta_i(u) = |N(u) \cap (U_i - Temp_i)|$$

Lemma 1. *Algorithm PA(G, t) executes at most* $2n$ *iterations of the while loop (i.e.,* $\lambda \leq 2n$*).*

Proof. First of all we prove that, at each iteration $i \geq 1$ of the while loop of PA(G, t), a node $v_i \in U_i$ is selected. If $R_i = \emptyset$ then $A_i \neq V$ (otherwise the algorithm terminates). Since by 1. of Fact 1 $V - U_i \subseteq A_i$ we have that there exist $u \in U_i$ such that $u \notin A_i$. Then using 2. of Fact 1 we have that $u \notin Temp_i$ and consequently $U_i - Temp_i - R_i \neq \emptyset$. Hence a node is selected by Case 1 or by Case 2 or at line 20 of the algorithm. Otherwise ($R_i \neq \emptyset$) and a node is selected by Case 1 or by Case 2 or at line 25 of the algorithm. We conclude the proof noticing each $v \in V$ can be selected at most twice: Once v is eventually inserted in $Temp$ (if Case 3 applies) and once v is removed from U (if either Case 1 or Case 2 apply). Indeed by 3. of Fact 1, Case 3 only applies to nodes in $U_i - Temp_i$.

Theorem 2. *For any graph* $G = (V, E)$ *and threshold function* $t(\cdot)$*, the algorithm* $PA(G,t)$ *returns a perfect seed set for* G *in* $O(|E| \log |V|)$ *time.*

Proof. In order to show that the set S provided by the algorithm PA(G,t) is a perfect seed set for G, we first show that for each $i = 1, \ldots, \lambda$ the set S_i is able to make all the nodes in

$$\mathcal{R}_i = \bigcup_{j=i}^{\lambda} (R_i \cup \{u \notin A_i \text{ such that } \delta_i(u) = 0\})$$

a spreader, that is $\mathcal{R}_i \subseteq \mathsf{Spreader}_{G[U_i]}[S_i]$. We show it by induction with i going from λ back to 1.

Consider first $i = \lambda$. Let v_λ a node in $G[U_\lambda]$. Since λ is the last step and at most one node is removed from R at each step, we have that $R_\lambda = \emptyset$ or $R_\lambda = \{v_\lambda\}$. We distinguish three cases on the selected node v_i.

- (Case 1 holds). In this case $k_\lambda(v_\lambda) = 0$ and v_λ is immediately spreader in $G[U_\lambda]$ and the statement is clearly satisfied.
- (Case 2 holds). In this case ($R_\lambda = \{v_\lambda\}$ and $k_\lambda(v_\lambda) > \delta_\lambda(v_\lambda)$) or ($v_\lambda \notin A_\lambda$ and $\delta_\lambda(v_\lambda) = 0$) and consequently $S_\lambda = \{v_\lambda\}$ and $\mathcal{R}_\lambda \subseteq \mathsf{Spreader}_{G[U_\lambda]}[S_\lambda]$.
- Finally we show that case 3 cannot hold at the last iteration of the algorithm. Indeed if $R_\lambda = \emptyset$ then $v_\lambda \notin A_\lambda$ (otherwise the algorithm cannot terminate at round λ). In this case a new node is added to R at the line 22 of the algorithm and the algorithm cannot terminate at round λ. We notice that this node must exists, otherwise $\delta_\lambda(v_\lambda) = 0$ and Case 2 holds. On the other hand, if $R_\lambda = \{v_\lambda\}$ then $U_\lambda - Temp_\lambda - R_\lambda = \emptyset$ and we have $U_\lambda - Temp_\lambda = \{v_\lambda\}$ and consequently $\delta_\lambda(v_\lambda) = 0$. Since we are in case tree we also know that $k_\lambda(v_\lambda) > 0$ and Case 2 holds.

Consider now $i < \lambda$ and suppose the algorithm be correct on $G[U_{i+1}]$, that is, $\mathcal{R}_{i+1} \subseteq \mathsf{Spreader}_{G[U_{i+1}]}[S_{i+1}]$. We show that the algorithm is correct on $G[U_i]$ with thresholds $k_i(u)$ for $u \in U_i$.

By the algorithm PA, for each $u \in U_i$ we have

$$k_{i+1}(u) = \begin{cases} \max(k_i(u) - 1, 0) & \text{if Case 1 or 2 hold and } u \in N(v_i) \cap U_i \\ k_i(u) & \text{otherwise,} \end{cases} \quad (1)$$

where v_i is the node selected at iteration i.

We distinguish three cases on the selected node v_i.

- (Case 3 holds). In this case $U_i = U_{i+1}$ and $S_{i+1} = S_i$. Moreover by (1), $k_{i+1}(u) = k_i(u)$ for each $u \in U_{i+1}$. If $v_i \notin R_i$ then $R_i \subseteq R_{i+1}$ and consequently $\mathcal{R}_i = \mathcal{R}_{i+1} \subseteq \mathsf{Spreader}_{G[U_{i+1}]}[S_{i+1}] = \mathsf{Spreader}_{G[U_i]}[S_i]$.
  Otherwise ($v_i \in R_i$) we have $R_i \subseteq R_{i+1} \cup \{v_i\}$, $U_i - Temp_i - R_i = \emptyset$ and by 3. of Fact 1 , we have $U_i - Temp_i = R_i$. Hence,
  $$(N(v) \cap (U_i - Temp_i)) \subseteq R_{i+1} \tag{2}$$
  Since we are in Case 3 and $v_i \in R_i$ then $\delta_i(v) \geq k_i(v)$. Using this, Fact 2 and equation (2), we have that since $\mathcal{R}_{i+1} \subseteq \mathsf{Spreader}_{G[U_{i+1}]}[S_{i+1}]$ then $S_{i+1} = S_i$ is able to make v_i a spreader in $G[U_i]$ and we have $\mathcal{R}_i \subseteq \mathsf{Spreader}_{G[U_i]}[S_i]$.
- (Case 2 holds). In this case $U_{i+1} = U_i - \{v_i\}$, $R_i \subseteq R_{i+1} \cup \{v_i\}$ and $S_i = S_{i+1} \cup \{v_i\}$. Hence $v_i \in \mathsf{Spreader}[S_i]$. Moreover by (1), it follows that for any $u \in N(v_i) \cap U_i$, if $u \in \mathsf{Spreader}[S_{i+1}]$ then $u \in \mathsf{Spreader}[S_i]$. Hence $\mathcal{R}_i \subseteq \mathsf{Spreader}[S_i]$.
- (Case 1 holds). In this case we have $k_i(v_i) = 0$, $U_{i+1} = U_i - \{v_i\}$, $R_i \subseteq R_{i+1} \cup \{v_i\}$ and $S_i = S_{i+1}$. Since $k_i(v_i) = 0$, node v_i is immediately spreader in $G[U_i]$. Hence by (1), each neighbor u of v_i in $G[U_i]$ is influenced by v_i and its threshold is updated according to (1). Therefore, since $\mathcal{R}_{i+1} \subseteq \mathsf{Spreader}_{G[U_{i+1}]}[S_{i+1}]$, we have that $\mathcal{R}_i \subseteq \mathsf{Spreader}_{G[U_i]}[S_i]$.

The statement follows since $G[U_1] = G$.

The theorem follows by observing that a node is moved to the set A only if $(v \cup N(v)) \cap \mathcal{R}_1 \neq \emptyset$ and that the algorithm terminates when all nodes are aware ($A = V$) and the set R is empty.

The PA algorithm can be implemented to run in $O(|E| \log |V|)$ time. Indeed we need to process the nodes $v \in V$—each one at most two times (see Lemma 1)—according to the metrics $\delta(v)$ and $k(v)/(\delta(v)(\delta(v)+1))$, and the updates, that follows each processed node $v \in V$ involve at most $|N(v)|$ neighbors of v.

4 Experimental Results

Due to Theorem 1, we cannot aim to any significant performance guaranteed on the seed set size for *general* graphs and threshold functions. Nonetheless, extensive experiments show that our algorithm performs very well on large real networks, both in terms of efficiency of the solution and of the running time.

We conducted experiments on 12 real networks of various sizes from the Stanford Large Network Data set Collection (SNAP) [24], the Social Computing Data Repository at Arizona State University [31] and Newman's Network data [26]. The main characteristics of the studied networks are shown in Table 1.

The active information diffusion problem is a novel model of information diffusion and, to the best of our knowledge, no heuristic is known for the PA problem. For this reason we decided to evaluate the effectiveness of our algorithm (PA) with two heuristics that respectively solve two problems related to the PA

Name	# of nodes	# of edges	Max degree	Size of the LCC	Clust. Coeff.	Modularity
BlogCatalog3 [31]	10312	333983	3992	10312	0.4756	0.2374
Ca-AstroPh [24]	18772	198110	504	17903	0.6768	0.3072
Ca-CondMath [24]	23133	93497	279	21363	0.7058	0.5809
Ca-GrQc [24]	5242	14496	81	4158	0.6865	0.7433
Ca-HepPh [24]	10008	118521	491	11204	0.6115	0.5085
Ca-HepTh [24]	9877	25998	65	8638	0.5994	0.6128
Cit-HepTh [24]	27770	352807	64	24700	0.3120	0.7203
Douban [31]	154907	327162	287	154908	0.048	0.5773
Facebook [24]	4039	88234	1045	4039	0.6055	0.8093
Jazz [26]	198	2742	100	198	17899	0.6334
Karate [26]	34	78	17	5	45	0.5879
Power grid [26]	4941	6594	19	4941	0.1065	0.6105

Table 1. The networks.

problem. The first heuristic, named *MTS* [15], is devoted to the minimum target set selection (TSS) problem where the aim is to have each node become a spreader. We have chosen this TSS heuristics since it experimentally outperforms the other known algorithms [13,22,29] for the TSS problem, see [15].

The rationale of this comparison is to show that by relaxing the goal of the TSS model for the new model (which only aims to make each node aware) we are able to identify significantly smaller seed sets.
On the other hand, when all the thresholds $t(v)$ are equal to the node degrees $d(v)$, the PA problem is equivalent to the well known Dominating Set problem. For this reason we will compare our algorithm with the (best known) heuristic [4], named *DOM*, for the Dominating Set problem.
Thresholds values. We tested the three algorithms using two categories of threshold function:

- *Random thresholds* where $t(v)$ is chosen uniformly at random in the interval $[1, d(v)]$. Since the random thresholds test settings involve some randomization, we executed each test 10 times. The results were compared using means of target set sizes (the observed variance was negligible);
- *Proportional thresholds,* where for each v the threshold $t(v)$ is set as $\alpha \times d(v)$ with $\alpha = 0.1, 0.2, \ldots, 1$. Notice that for $\alpha = 0.5$ we are considering a particular version of the activation process named "majority" thresholds, while for $\alpha = 1$ we are considering the Dominating Set problem.

4.1 Test Results

Random Thresholds. Table 2 depicts the results of the Random threshold test setting. Each number represents the average size of the perfect seed set generated by PA and MTS algorithms on each network using random thresholds (for each test setting, the same thresholds values have been used for both the algorithms). The

Name	PA	MTS
BlogCatalog3	10	12 (20%)
Ca-AstroPh	919	1157 (25.9%)
Ca-CondMath	1573	1810 (15.07%)
Ca-GrQc	636	661 (3.93%)
Ca-HepPh	790	901 (14.05%)
Ca-HepTh	964	945 (-1.97%)
Cit-HepTh	955	1045 (9.42%)
Douban	2374	2343 (-1.31%)
Facebook	9	213 (2267%)
Jazz	4	7 (75%)
Karate	3	3 (0%)
Power grid	352	340 (-3.41%)

Table 2. Random Thresholds Results: For each network and each algorithm, the average size of the perfect seed set is depicted.

value in bracket represents the overhead percentage of MTS algorithm compared to the PA algorithm.

Constant and Proportional thresholds. Figures 1 and 2 report the results for the proportional thresholds settings. For each network the plot depicts the size of the perfect seed set (Y-axis), for each value of $\alpha \in [0.1, 1]$ (X-axis) and for each algorithm (series). We present the results only for 4 networks because of space limits; the experiments performed on the other networks exhibit similar behaviors.
The results in Fig. 1 and 2 confirm our hypothesis. The size of the initial seed set provided by our PA algorithm is in general significantly smaller than the size of the set provided by the other strategies. We notice that the gap between the PA and the MTS algorithms increase with the value of the node thresholds (this result was expected: the larger the value of $t()$, the larger the difference between the models). The PA algorithm is always better than the DOM algorithm, when $t(v) < d(v)$. Moreover when $t(v) = d(v)$ (that is, when the PA problems becomes the Dominating Set Problem), the two algorithms provide comparable results, hence the PA algorithm could be considered as an effective alternative heuristics for the dominating set problem.

5 Trees.

Let $T = (V, E)$ be a tree rooted at any node r, and let $T(v)$ the subtree rooted at v, for any $v \in V$. We can prove that the algorithm PA outputs an optimal perfect seed set whenever the input graph is a tree.

Theorem 3. *PA(T, t) returns an* optimal *perfect seed set for any tree T.*

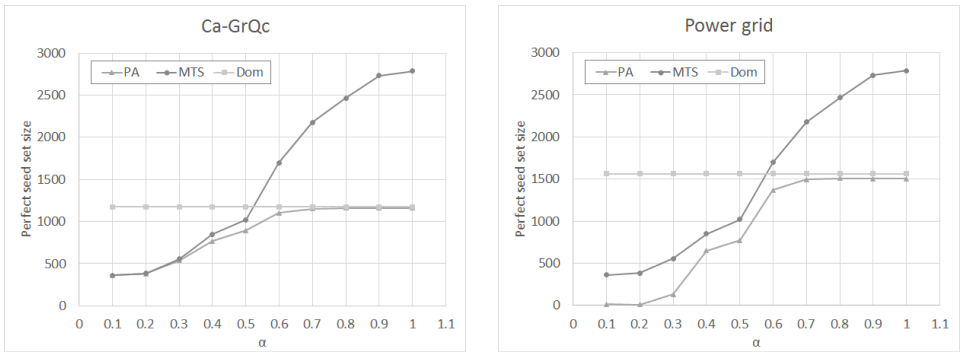


Fig. 1. Proportional Thresholds Results: CA-GrQc network and Power grid network

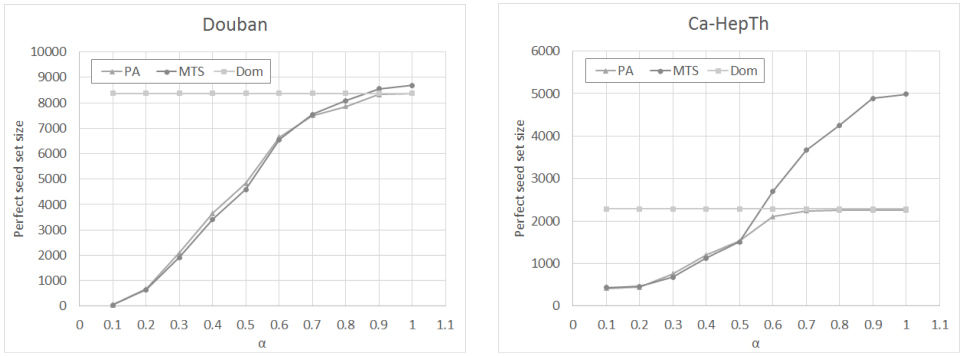


Fig. 2. Proportional Thresholds Results: Douban network and Ca-HepTh network.

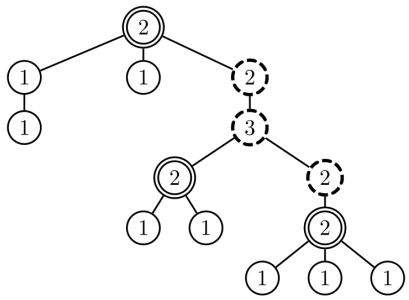


Fig. 3. Numbers inside circles are the node thresholds, double–circled denote seeds, dashed–circled lines denote aware nodes, solid–circled nodes denote spreaders.

If T is the tree in Fig. 3 one can see that the algorithm PA(T, t) returns a optimal seed set—consisting of the three double-circled nodes in the figure.

In order to evaluate the time complexity for trees, we report as TREE-PA the rewriting of the general PA algorithm in Section 3 in case the input graph is known to be a tree. One can see that the algorithm essentially computes the seed set while performing a visit (in BFS reverse order) of the tree. We can then show that

Theorem 4. *The* PA *problem can be solved in linear time for any tree.*

Algorithm 2: TREE-PA(T, t), $T = (V, E)$ is a tree with thresholds $t(v)$ for $v \in V$

```
1  S = ∅;   A = ∅;   P = ∅;
2  foreach v ∈ V in a BFS reverse order do
3      if v ≠ r then                                   // v is not the root node
4          if t(v) = 0 then
5              t(f_v) = t(f_v) − 1;                    // f_v denotes v's father
6              A = A ∪ {f_v}
7          else
8              if v ∈ P AND t(v) ≥ 2 then
9                  S = S ∪ {v};
10                 t(f_v) = t(f_v) − 1;
11                 A = A ∪ {f_v}
12             else
13                 if v ∉ A OR (v ∈ P AND t(v) = 1) then
14                     P = P ∪ {f_v}                   // f_v must spread
15     if v = r AND t(v) > 0 AND v ∉ A − P then S = S ∪ {v}
16 return S
```

6 Conclusion and Open Problems

We have studied some algorithmic aspects of a recently introduced information diffusion model, that differentiates among spreaders and aware nodes [14]. Many interesting questions related to this model remain open and might be interesting to study:

- Real life social networks are characterized by the existence of highly connected communities and it was observed that in real networks, having high modularity [25], it is often difficult for information to flow from one community to another. This suggests that one should consider each (dense) community separately. From a result in [14], we know that it is possible to relate the minimum graph degree to the size of a perfect seed set. Namely, in any graph G with $t(v) \leq t$ and $d(v) \geq \frac{|V|+t-3}{2}$, for each $v \in V$, any independent set which is either maximal or has size $2t - 2$ is a perfect seed set for G. Establishing a significant lower

bound on the size of the seed set of a dense graph has (so far) eluded our efforts. However, we recall that deciding if there exists a perfect seed set of size less than t is a hard problem in general. It would be interesting to establish to what extent such an hardness result still holds for dense graphs.

- More generally, are there class of graphs, other then trees and cliques, for which the problem can be either efficiently solved or admits a *small* approximation factor?
- It would also be interesting to determine a significant upper bound on the size of a perfect seed set in terms of node degree and threshold, in the spirit of the bound derived in [1] for the TSS problem.

References

1. Eyal Ackerman, Oren Ben-Zwi, and Guy Wolfovitz. Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, 411(44–46):4017–4022, 2010.
2. Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The role of social networks in information diffusion. In *Proceedings of the 21st International Conference on World Wide Web*, pages 519–528, 2012.
3. Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87–96, 2011.
4. Alina Campan, Traian Marius Truta, and Matthew Beckerich. Fast dominating set algorithms for social networks. In *MAICS*, 2015.
5. Carmen C. Centeno, Mitre C. Dourado, Lucia Draque Penso, Dieter Rautenbach, and Jayme L. Szwarcfiter. Irreversible conversion of graphs. *Theoretical Computer Science*, 412(29):3693–3700, 2011.
6. Ning Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.
7. Wei Chen, Carlos Castillo, and Laks Lakshmanan. *Information and Influence Propagation in Social Networks*. Morgan & Claypool, 2013.
8. Chun-Ying Chiang, Liang-Hao Huang, and Hong-Gwa Yeh. Target set selection problem for honeycomb networks. *SIAM Journal on Discrete Mathematics*, 27(1):310–328, 2013.
9. Morgan Chopin, André Nichterlein, Rolf Niedermeier, and Mathias Weller. Constant thresholds can make target set selection tractable. *Theory of Computing Systems*, 55(1):61–83, 2014.
10. Ferdinando Cicalese, Gennaro Cordasco, Luisa Gargano, Martin Milanič, Joseph Peters, and Ugo Vaccaro. Spread of influence in weighted networks under time and budget constraints. *Theoretical Computer Science*, 586:40–58, 2015.
11. Ferdinando Cicalese, Gennaro Cordasco, Luisa Gargano, Martin Milanič, and Ugo Vaccaro. Latency-bounded target set selection in social networks. *Theoretical Computer Science*, 535:1 – 15, 2014.
12. Amin Coja-Oghlan, Uriel Feige, Michael Krivelevich, and Daniel Reichman. Contagious sets in expanders. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1953–1987, 2015.
13. Gennaro Cordasco, Luisa Gargano, Marco Mecchia, Adele A. Rescigno, and Ugo Vaccaro. A fast and effective heuristic for discovering small target sets in social networks. In *Proc. of COCOA 2015*, volume 9486, pages 193–208, 2015.

14. Gennaro Cordasco, Luisa Gargano, Adele A. Rescigno, and Ugo Vaccaro. Evangelism in social networks. In *proceedings of 27th International Workshop on Combinatorial Algorithms (To Appear)*, 2016.
15. Gennaro Cordasco, Luisa Gargano, and Adele Anna Rescigno. Influence propagation over large scale social networks. In *Proceedings of ASONAM 2015, Paris, France*, pages 1531–1538, 2015.
16. Thang N. Dinh, Huiyuan Zhang, Dzung T. Nguyen, and My T. Thai. Cost-effective viral marketing for time-critical campaigns in large-scale social networks. *IEEE/ACM Trans. Netw.*, 22(6):2001–2011, December 2014.
17. David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.
18. Luisa Gargano, Pavol Hell, Joseph G. Peters, Ugo Vaccaro. Influence diffusion in social networks under time window constraints. *Theor. Comput. Sci.*, 584(C):53–66, 2015.
19. David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. of the ACM SIGKDD KDD 2003*, pages 137–146, 2003.
20. David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proc. of ICALP 2005*, pages 1127–1138, 2005.
21. David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11(4):105–147, 2015.
22. Suman Kundu and Sankar K. Pal. Deprecation based greedy strategy for target set selection in large scale social networks. *Information Sciences*, 316:107–122, 2015.
23. Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Trans. Web*, 1(1), May 2007.
24. Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2015.
25. Mark E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. U.S.A.* 103 (23): 8577âĂŞ8582.
26. Mark Newman. Network data, `http://www-personal.umich.edu/~mejn/netdata/`, 2015.
27. André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, Mathias Weller. On tractable cases of target set selection. *Social Network Analysis and Mining*, 3(2):233–256, 2013.
28. T. V. Thirumala Reddy and C. Pandu Rangan. Variants of spreading messages. *J. Graph Algorithms Appl.*, 15(5):683–699, 2011.
29. Paulo Shakarian, Sean Eyre, and Damon Paulo. A scalable heuristic for viral marketing under the tipping model. *Social Network Analysis and Mining*, 3(4):1225–1248, 2013.
30. Michela Del Vicario, Alessandro Bessi, Fabiana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, Eugene Stanley and Walter Quattrociocchi. The spreading of misinformation onlines. *PNAS*, 113(3):554–559, 2016.
31. Reza Zafarani and Huan Liu. Social computing data repository at ASU. http://socialcomputing.asu.edu, 2009.

The cost of securing IoT communications*

Chiara Bodei and Letterio Galletta

Dipartimento di Informatica, Università di Pisa
{chiara,galletta}@di.unipi.it

Abstract. More smart objects and more applications on the Internet of Things (IoT) mean more security challenges. In IoT security is crucial but difficult to obtain. On the one hand the usual trade-off between highly secure and usable systems is more impelling than ever; on the other hand security is considered a feature that has a cost often unaffordable. To relieve this kind of problems, IoT designers not only need tools to assess possible risks and to study countermeasures, but also methodologies to estimate their costs. Here, we present a preliminary methodology, based on the process calculus IoT-LySA, to infer quantitative measures on systems evolution. The derived quantitative evaluation is exploited to establish the cost of the possible security countermeasures.

1 Introduction

Within few years the objects we use every day will have computational capabilities and will be always connected to the Internet. In this scenario, called the Internet of Things (IoT), these "smart" devices are equipped with sensors to automatically collect different pieces of information, store them on the cloud or use them to affect the surrounding environment through actuators. For instance, our smart alarm clock can drive our heating system to prepare us a warm bathroom in the morning, while an alarm sensor in our place can directly trigger an emergency call to the closest police station. Also, in a storehouse stocking perishable food, equipped with sensors to determine the internal temperature and other relevant attributes, the refrigeration system can automatically adapt the temperature according to the information collected by sensors.

The IoT paradigm introduces new pressing security challenges. On the one hand, the usual trade-off between highly secure and usable systems is more critical than ever. On the other hand, security is considered a costly feature for devices with limited computational capabilities and with limited battery power.

Back to the refrigerator system above, an attacker can easily intercept sensors communications, manipulate and falsify data. We can resort to cryptography and

* Work supported by project PRA_2016_64 "Through the fog" (University of Pisa).

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 163–176 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

to consistency checks to prevent falsification and to detect anomalies. But is it affordable to secure all the communications? Can we still obtain a good level of security by protecting only part of communications?

IoT designers have to be selective, e.g. in choosing which packets to encrypt. To this aim, designers not only need tools to assess possible risks and to study countermeasures, but also methodologies to estimate their costs. The cost of security can be computed in terms of time overhead, energy consumption, bandwidth, and so on. All these factors must be carefully evaluated for achieving an acceptable balance among security, cost and usability of the system.

First, we introduce functions over the enhanced labels to associate costs to transitions. Here, the cost of a system is specified in term of the *time* spent for transitions, and it depends on the performed action as well as on the involved nodes. However, we can easily treat other quantitative properties, e.g. energy consumption. Intuitively, cost functions define exponential distributions, from which we compute the rates at which a system evolves and the corresponding CTMC. Then, to evaluate the performance we calculate the stationary distribution of the CTMC and the transition rewards.

Usually, formal methods provide designers with tools to support the development of systems and to reason about their properties, both qualitative and quantitative. Here, we present some preliminary steps towards the development of a formal methodology to support the analysis of the security cost in IoT systems. We aim at providing a general framework with a mechanisable procedure (with a small amount of manual tuning), where quantitative aspects are symbolically represented by parameters. Their instantiation is delayed until hardware architectures and cryptographic algorithms are fixed. By only changing these parameters designers could compare different implementations of an IoT system and could choose the better trade-off between security and costs.

Technically, we define an enhanced semantics for IoT-LySA, a process calculus recently proposed to model and reason about IoT systems [4]. IoT-LySA is equipped with a static analysis that safely approximates how data from sensors spread across the system and how objects interact each other's. Our enhanced semantics follows the methodology of [7], where each transition is associated to a cost in the style of [17,1]. Here, the cost is specified in term of the *time* spent for the transition i.e. it is the rate of the transition. From rates we mechanically derive a continuous-time Markov chains that can be analysed using standard techniques and tools [19,22]. For simplicity, here we consider a subset of IoT-LySA without actuators where intra-node communication is carried out through message passing instead of a shared store. Note that our approach can be extended to deal with other optimisation criteria, e.g. energy consumption, particularly critical in IoT systems.

Structure of the paper. In the next section, we briefly introduce the process calculus IoT-LySA. In Section 3, we present a simple function that assigns rates to transitions, and we show how to obtain the CTMC associated with a given system of nodes and how to extract performance measures from it. Concluding remarks and related work are in Section 4.

2 IoT-LySA and its Enhanced Semantics

In [4] an IoT system consists of a set of nodes that communicate through message-passing. Each node is uniquely identified by a label and is made of logical (processes) and physical (sensors and actuators) components that interact through a shared store. For simplicity, in the following we do not consider actuators, and we replace conditional construct with non deterministic choice. Furthermore, following the approach in [3] we assume a finite set of keys that are known a priori by the nodes.

Syntax. In IoT-LySA systems $N \in \mathcal{N}$ consist of a fixed number of nodes that host control processes $P \in \mathcal{P}$ and sensors $S \in \mathcal{S}$. The syntax is in Tab. 1, where $\mathcal{V}$ denotes the set of values, while $\mathcal{X}$ and $\mathcal{Z}$ are the local and the global variables, respectively, and $\mathcal{K} \subseteq \mathcal{V}$ denotes the set of cryptographic keys (e.g. K_0).

$\mathcal{N} \ni N ::=$ *systems of nodes*		$\mathcal{B} \ni B ::=$ *node components*	
$\mathsf{0}$	inactive node	P	process
$\ell : [B]$	single node	S	sensor
$N_1 \mid N_2$	composition	$B \parallel B$	composition

$P ::=$ *control processes*	
$\mathsf{0}$	nil
$\langle E_1, \cdots, E_k \rangle . P$	intra-node output
$\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L . P$	multi-output L$\subseteq \mathcal{L}$
$(E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k) . P$	input (with match.)
$P_1 + P_2$	summation
$A(y_1, \ldots, y_n)$	recursion
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k\}_{K_0}\ \mathsf{in}\ P$	decryption (with match.)
$(i; z_i).P$	clear input from sensor i
$(\{i; z_i\}_K).P$	crypto input from sensor i

$S ::=$ *sensor processes*		$E ::=$	*terms*
$\tau.S$	internal action	v	value
$\langle i, v \rangle . S$	i^{th} output	x	variable
$\langle \{i, v\}_K \rangle . S$	i^{th} enc. output	z	sensor's variable
$A(y_1, \ldots, y_n)$	recursion	$\{E_1, \cdots, E_k\}_{K_0}$	encryption
		$f(E_1, \cdots, E_n)$	function appl.

Table 1. Syntax of IoT-LySA.

In the syntax of systems, $\mathsf{0}$ denotes the null inactive system; a single node $\ell : [B]$ is uniquely identified by a label $\ell \in \mathcal{L}$ (which represents a node property such as location). Node components B are obtained by the parallel composition (through the operator $||$) of control processes P, and of a fixed number of sensors S. We assume that sensors are identified by an unique identifier $i \in \mathcal{I}_\ell$.

In the syntax of processes, $\mathsf{0}$ represents the *inactive* process (inactive components of a node are all coalesced). The process $\langle E_1, \cdots, E_k \rangle . P$ sends the tuple $E_1, \cdots, E_k$ to another process in the same node and then continues like P.

The process $\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L.P$ sends the tuple $E_1, \ldots, E_k$ to the nodes whose labels are in L and evolves as P. The process $(E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k).P$ receives a tuple $E'_1, \cdots, E'_k$: if the first j terms of the received message pairwise match the first j terms of the input tuple, the message is accepted, otherwise is discarded (see later for details). The operator $\|$ describes parallel composition of processes, while $+$ denotes non-deterministic choice. An agent is a static definition of a parameterised process. Each agent identifier A has a unique defining equation of the form $A(y_1, \ldots, y_n) = P$, where $y_1, \ldots, y_n$ are distinct names occurring free in P. The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{K_0}\ \mathsf{in}\ P$ tries to decrypt an encrypted value using the key K_0, provided that the first j elements of the decrypted term coincide with the terms E_j.

A sensor can perform an internal action τ or send an (encrypted) value v, gathered from the environment, to its controlling process and continues as S. We do not provide an explicit operation to read data from the environment but it can be easily implemented as an internal action.

Finally, in the syntax of term, a value represents a piece of data (e.g. keys or values read the environment). As said above, we have two kinds of disjoint variables: x are standard local variables, used as in π-calculus; while sensor variables z belong to a node and are globally accessible within it. As usual, we require that variables and names are disjoint. The encryption function $\{E_1, \cdots, E_k\}_{K_0}$ returns the result of encrypting values E_i for $i \in [1, k]$ with the key K_0. The term $f(E_1, \cdots, E_n)$ is the application of function f to n arguments; we assume given a set of primitive aggregation functions, e.g. functions for comparing values.

Working Example We set a simple IoT system up to keep the temperature under control inside a storehouse with perishable food (a big quadrangular room). We install four sensors, one for each corner of the storehouse. Each sensor S_i periodically senses (by means of the function $sense_i()$) the temperature and sends it through a wireless communication to a control unit P_c in the same node N_1. The unit P_c computes the average temperature and checks if it is within accepted bounds. If this is not the case, P_c sends an alarm through other nodes and the Cloud. We want to prevent an attacker from intercepting and manipulating data sent by sensors. A possible approach consists in exploiting the fact that sensors on the same side should sense the same temperature, with a difference that can be at most a given value ϵ. The control unit can easily detect anomalies and discard data tailored by an attacker, comparing values coming from the sensors that are on the same side of the room. But what happens if the attacker falsifies the data sent by more than one sensor? A possible solution consists in enabling some sensors (in our example S_1 and S_3) to use cryptography for obtaining reliable data. Nevertheless, before adopting this solution we would like to evaluate it by estimating its cost. The control process P_c in the node N_1 reads data from sensors, compares them and compute their average and sends them to the node N_2. The process Q_c of N_2 sends an `alarm` or an `ok` message to the node N_3 depending on the received data, together with the average temperature. The process R_c of N_3 is an Internet service that waits for messages from N_2 and

handles them (through the internal action τ). The IoT-LySA specification of the storehouse system follows:

$$
\begin{array}{lll}
N & = N_1 \mid N_2 \mid N_3 = \ell_1 : [P_c \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \ell_2 : [Q_c \parallel \mathtt{0}] \mid \ell_3 : [R_c \parallel \mathtt{0}] \\
P_c & = (0;\ z_0).\,\tau.(\{1; z_1\}_{K_1}).\tau.(2;\ z_2).\,\tau.(\{3; z_3\}_{K_3}).\tau. \\
 & \quad \langle\!\langle cmp(z_0, ..., z_3), avg(z_0, ..., z_3) \rangle\!\rangle \triangleright \{\ell_2\}.\tau.P_c \\
Q_c & = (\mathtt{true}; x_{avg}).\langle\!\langle \mathtt{ok}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c + (\mathtt{false}; x_{avg}).\langle\!\langle \mathtt{alarm}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c \\
R_c & = (; w_{res}, w_{avg}).\tau.R_c \\
S_m & = \langle m, sense_m() \rangle.\,\tau.S_m \qquad m = 0, 2 \\
S_j & = \langle \{j, sense_j()\}_{K_j} \rangle.\,\tau.S_j \qquad j = 1, 3
\end{array}
$$

The function *cmp* performs consistency checks, by *comparing* data coming from insecure sensors with data from secure ones: it returns `true` if data are within the established bounds, `false` otherwise. The function *avg* computes the average of its arguments. We suppose that processes and sensors perform some internal activities (τ-actions). Another possible solution consists in having just one sensor that uses cryptography. This new system of nodes $\hat{N}$ differs from the previous one in the specification of the process $\hat{P}_c$ in the first node:

$$
\begin{array}{l}
\hat{P}_c = (0;\ z_0).\,\tau.(\{1; z_1\}_{K_1}).\tau.(2;\ z_2).\,\tau.(3;\ z_3).\,\tau. \\
\qquad \langle\!\langle halfcmp(z_0, z_1), avg(z_0, ..., z_3) \rangle\!\rangle \triangleright \{\ell_2\}.\tau.\hat{P}_c
\end{array}
$$

where the comparison function *halfcmp* uses only two arguments. We expect that this second solution is less expensive, and we apply our methodology to formally compare the relative costs of the two solutions.

Enhanced Operational Semantics. To estimate cost, we give an *enhanced* reduction semantics following [2,6,7]. The underlying idea is that each transition is enriched with an *enhanced label* θ, which records information about the transition. Actually, we label transitions for communications and decryptions. For communications, we record the action (input or output) with the corresponding prefixes, and the labels of the involved nodes. For decryption, we store the label of the node performing the operation and information about the data. Note that in the following we use the abbreviations *out*, *in*, *dec*, for denoting the communication prefixes, the decryption constructs and the possible function calls f inside them. We can obtain a standard semantics by simply removing the labels.

Definition 1. *Given* $\ell, \ell_O, \ell_I, \ell_D \in \mathcal{L}$, enhanced labels *theta are defined as:*

$$
\begin{array}{lll}
\Theta \ni \theta ::= & \langle \ell\ \{out\}, \ell\ \{in\} \rangle & \textit{internal secure communication} \\
 & \langle \ell\ out, \ell\ in \rangle & \textit{internal communication} \\
 & \langle \ell_O\ out, \ell_I\ in \rangle & \textit{inter-nodes communication} \\
 & \{\ell_D\ dec\} & \textit{decryption of a message}
\end{array}
$$

As usual, our semantics consists of the standard structural congruence $\equiv$ on nodes, processes and sensors and of a set of rules defining the transition relation. Our notion of *structural congruence* $\equiv$ is standard except for the following congruence rule for processes that equates a multi-output with empty set of receivers to the inactive process: $\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright \emptyset.\mathsf{0} \equiv \mathsf{0}$.

Our *reduction relation* $\xrightarrow{\theta} \subseteq \mathcal{N} \times \mathcal{N}$ is defined as the least relation on closed nodes, processes and sensors that satisfies a set of inference rules. Our rules are quite standard apart from the five rules for communications shown in Tab. 2 and briefly commented below. We assume the standard meaning for terms $[\![E]\!]$.

(Sensor-Com)

$$\ell : [\langle i, v_i \rangle.\, S_i \parallel (i;\ z_i).\, P \parallel B] \xrightarrow{\langle \ell\ out, \ell\ in \rangle} \ell : [S_i \parallel P\{v_i/z_i\} \parallel B\{v_i/z_i\}]$$

(Crypto-Sensor-Com)

$$\ell : [\langle \{i, v_i\}_K \rangle.\, S_i \parallel (\{i; z_i\}_K).P \parallel B] \xrightarrow{\langle \ell\ \{out\}, \ell\ \{in\} \rangle} \ell : [S_i \parallel P\{v_i/z_i\} \parallel B\{v_i/z_i\}]$$

(Intra-Com)

$$\frac{\bigwedge_{i=1}^{k} v_i = [\![E_i]\!] \ \wedge\ \bigwedge_{i=1}^{j} [\![E_i]\!] = [\![E'_i]\!]}{\ell : [\langle E_1, ..., E_k \rangle.\, P \parallel (E'_1, ..., E'_j; x_{j+1}, ..., x_k).Q \parallel B] \xrightarrow[\langle \ell\ out, \ell\ in \rangle]{} \ell : [P \parallel Q\{v_{j+1}/x_{j+1}, ..., v_k/x_k\} \parallel B]}$$

(Multi-Com)

$$\frac{\ell_2 \in L \ \wedge\ Comp(\ell_1, \ell_2) \ \wedge\ \bigwedge_{i=1}^{k} v_i = [\![E_i]\!] \ \wedge\ \bigwedge_{i=1}^{j} [\![E_i]\!] = [\![E'_i]\!]}{\ell_1 : [\langle\!\langle E_1, ..., E_k \rangle\!\rangle \triangleright L.P_{11} \parallel B_P] \mid \ell_2 : [(E'_1, ..., E'_j; x_{j+1}, ..., x_k).Q_{11} \parallel B_Q] \xrightarrow[\langle \ell_1\ out, \ell_2\ in \rangle]{} \ell_1 : [\langle\!\langle E_1, ..., E_k \rangle\!\rangle \triangleright L \setminus \{\ell_2\}.P_{11} \parallel B_P] \mid \ell_2 : [Q_{11}\{v_{j+1}/x_{j+1}, ..., v_k/x_k\} \parallel B_Q]}$$

Table 2. Operational semantic rules for communication.

The rule (*Sens-Com*) is for communications among sensors and processes: the variables used in the input are assumed *global* inside the node, in such a way that sensors are considered as a shared data structure $z_1, \cdots, z_n$. Therefore, the substitution is performed in all the processes of the node. The rule (*Crypto-Sens-Com*) is similar but it also requires that the receiving process successfully decrypts the encrypted data sent by a sensor. The rule (*Intra-Com*) is for intra-node communications. This construct implements also a matching feature: the communication succeeds, as long as the first j values of the message match the evaluations of the first j terms in the input. If this is the case, the result of evaluating each E_i is bound to each x_i.

The rule (*Multi-Com*) implements the inter nodes communication: the communication between the node labelled ℓ_1 and the node ℓ_2 succeeds, provided that (i) ℓ_2 is in the set L of receivers, (ii) the two nodes are compatible according to the compatibility function $Comp$, and (iii) the matching mechanism succeeds. If this is the case, the sender removes ℓ' from the set of receivers L, while in the second node, the receiving process continues bounding the result of each E_i to each variable x_i. Outputs terminate when all the nodes in L have received the message (see the congruence rule). Note that point-to-point communication amounts to

the case in which L is a singleton. The compatibility function $Comp$ defined over node labels is used to model constraints on communication, e.g. proximity, with $Comp(\ell_1, \ell_2)$ that yields true only when the two nodes ℓ_1, ℓ_2 are in the same transmission range. Of course, this function could be enriched for considering other notions of compatibility.

Hereafter, we assume the standard notion of transition system. Intuitively, it is a graph, in which systems of nodes form the nodes and (labelled) arcs represent the possible transitions between them. As will be clearer in the next section, we will only consider *finite* state systems, because finite states have an easier stochastic analysis. Note that this does not mean that the behaviour of such processes is finite, because their transition systems may have loops.

Example (cont'd) Back to our example, consider the following run of the first system where, for brevity, we ignored their internal actions τ

$$N \xrightarrow{\theta_0} N' \xrightarrow{\theta_1} N'' \xrightarrow{\theta_2} N''' \xrightarrow{\theta_3} N'''' \xrightarrow{\theta_{4i}} \begin{cases} N_0'''''' \xrightarrow{\theta_{50}} N \text{ if } i = 0 \\ N_1'''''' \xrightarrow{\theta_{51}} N \text{ if } i = 1 \end{cases}$$

the systems of nodes $N', N'', N''', N'''', N''''''$ are the intermediate ones reached from N during the computation and the labels θ_j annotate the j^{th} transition (θ_{ji} depending on the branch of the summation). In the run, fully specified below, the sensors of N_1 send a message to the process P_c, which checks the received data and sends the checking result to N_2. We denote with P_c' (Q_c', R_c', resp.) the continuations of P_c (Q_c, R_c, resp.) after the first input prefixes, with v_{comp} the value $cmp(v_0, ..., v_3)$, with v_{avg} the value $avg(v_0, ..., v_3)$, and with v_{res_i} (with $i = 0, 1$) the value `ok` (`alarm` respectively), depending on which branch of the summation is chosen. The evolution of the second system $\hat{N}$ is analogous to the one of N: the transition labels are such that $\hat{\theta}_{4i} = \langle \ell_1 \langle \langle halfcmp(v_0, v_1), avg(v_0, \cdots, v_3) \rangle \rangle, \ell_2(v_{bool}; x_{avg}) \rangle$ and $\hat{\theta}_l = \theta_l$ for $l \neq 4i$ (the transition labels θ_l are presented below, after the run of N).

$$\begin{aligned}
N \quad &= \ell_1 : [(0;\ z_0).\, P_c' \parallel P \parallel (\langle 0, sense_0() \rangle.\, \tau.S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 \xrightarrow{\theta_0} \\
N' \quad &= \ell_1 : [P_c'\{0/z_0\} \parallel (\tau.S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 \xrightarrow{\theta_1}\xrightarrow{\theta_2}\xrightarrow{\theta_3} \\
N'''' \quad &= \ell_1 : [P_c'\{0/z_0, 1/z_1, 2/z_2, 3/z_3\} \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 = \\
&\quad \ell_1 : [\langle\!\langle v_{comp}, v_{avg} \rangle\!\rangle \triangleright \{\ell_2\}.\tau.P_c \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
&\quad \ell_2 : [(\texttt{true}; x_{avg}).\langle\!\langle \texttt{ok}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c + \\
&\qquad (\texttt{false}; x_{avg}).\langle\!\langle \texttt{alarm}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c] \mid N_3 \xrightarrow{\theta_{4i}} \\
N_i'''''' \quad &= \ell_1 : [P_c \parallel P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
&\quad \ell_2 : [Q_c'\{v_{avg}/x_{avg}\}) \parallel 0] \mid \ell_3 : [(; w_{res}, w_{avg}).\tau.R_c \parallel 0] = \\
&\quad \ell_1 : [P_c \parallel P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
&\quad \ell_2 : [\langle\!\langle v_{res_i}, v_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c \parallel 0] \mid \ell_3 : [(; w_{res}, w_{avg}).\tau.R_c \parallel 0] \xrightarrow{\theta_{5i}} \\
N \quad &= \ell_1 : [P_c \parallel P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \ell_2 : [Q_c \parallel 0] \mid \ell_3 : [R_c'\{v_{res_i}/w_{res}, v_{avg}/w_{avg}\} \parallel 0]
\end{aligned}$$

$$\begin{aligned}
&\theta_0 = \theta_2 = \langle \ell_1 \langle j, v_j \rangle, \ell_1(j; z_i) \rangle \\
&\theta_1 = \theta_3 = \langle \ell_1 \langle \{j, v_j\}_{K_i} \rangle, \ell_1(\{j; z_j\}_{K_i}) \rangle \\
&\theta_{4i} = \langle \ell_1 \langle \langle cmp(v_0, \cdots, v_3), avg(v_0, \cdots, v_3) \rangle \rangle, \ell_2(v_{bool}; x_{avg}) \rangle \\
&\theta_{5i} = \langle \ell_2\ \langle \langle v_{res_i}, v_{avg} \rangle \rangle, \ell_3(w_{res}, w_{avg}) \rangle
\end{aligned}$$

3 Stochastic Semantics

We now show how to generate a Continuous Time Markov Chains (CTMC) from a transition system (see [17] for more details). First, we introduce functions over the enhanced labels to associate costs to transitions. Here, the cost of a system is specified in term of the *time* spent for transitions, and it depends on the performed action as well as on the involved nodes. However, we can easily treat other quantitative properties, e.g. energy consumption. Intuitively, cost functions define exponential distributions, from which we compute the rates at which a system evolves and the corresponding CTMC. Then, to evaluate the performance we calculate the stationary distribution of the CTMC and the transition rewards.

Cost Functions. Our cost functions assign a *rate* to each transition with label ϑ. To define this rate, we suppose to execute each action on a dedicated architecture that only performs that action, and we estimate the corresponding duration. To model the performance degradation due to the run-time support, we introduce a scaling factor for r for each routine called by the implementation under consideration. Here, we just propose a format for these functions, with parameters that depend on the nodes to be instantiated on need. For instance, in a node where the cryptographic operations are performed at very high speed (e.g. by a cryptographic accelerator), but with a slow link (low bandwidth), the time will be low for encryptions and high for communication; vice versa, in a node offering a high bandwidth, but poor cryptography resources.

Technically, we interpret costs as parameters of exponential distributions $F(t) = 1 - e^{-rt}$, with *rate* r and t as time parameter (general distributions are also possible see [18]): the transition rate r is the parameter that identifies the exponential distribution of the duration times of the transition, as usual in stochastic process algebras (e.g. [8]). The shape of $F(t)$ is a curve that grows from 0 asymptotically approaching 1 for positive values of its argument t. The parameter r determines the slope of the curve: the greater r, the faster $F(t)$ approaches its asymptotic value. The exponential distributions that we use enjoy the *memoryless property*, i.e. the occurrence of a new transition does not depend on the previous ones. We also assume that transitions are *time homogeneous* (the transitions enabled in a given state cannot be disabled by the flow of time).

We define in a few steps the function that associates rates with the (enhanced labels of) communication and decryption transitions. For the sake of simplicity, we ignore the costs for other primitives, e.g. constant invocation, parallel composition, summation, τ actions (see [17] for a complete treatment); we further neglect the sensor cost of sensing from the environment We need the auxiliary function $f_E : \mathcal{E} \to \mathbb{R}^+$ that estimates the effort needed to manipulate terms.

- $f_E(v) = size(v)$
- $f_E(\{E_1, \ldots, E_k\}_{K_0}) = f_{enc}(f_E(E_1), ..., f_E(E_1), crypto\_system, kind(K_0))$

The size of a value v matters. For an encrypted term, we use the function f_{enc}, which depends on the terms to encrypt, on the used crypto-system and on the

kind (short/long, short-term/long-term) of the key. The function $\$_\alpha : \mathcal{A} \to \mathbb{R}^+$ assigns costs to I/O and decryption prefix actions $\alpha \in \mathcal{A}$.

- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = f_{out}(f_E(E_1), ..., f_E(E_1), bw)$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = f_{in}(f_E(E_1), ..., f_E(E_j), match(j), bw)$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}) =$
  $f_{dec}(f_E(E), crypto\_system, kind(K_0), match(j))$

the send and receive primitives. Besides the implementation cost due to their own algorithms, the functions above depend on the bandwidth of the channel (represented by bw), on the cost of the exchanged terms (computed by f_E), and on the nodes involved in the communication. Moreover, the inter-node communication depends on the proximity-relation (e.g. the transmission range between nodes), represented here by the function $f_{<>}(\ell_O, \ell_I)$. Also, the cost of an input depends on the number of required matchings (represented by $match(j)$). Finally, the function f_{dec} represents the cost of a decryption, whose cost is similar to the one for encryption, with the additional cost of matchings. Finally, the function $\$: \Theta \to \mathbb{R}^+$ associates rates with enhanced labels.

- $\$(\langle \ell_O\ out, \ell_I\ in \rangle) = f_{<>}(\ell_O, \ell_I) \cdot \min\{\$_\alpha(out, \ell_O), \$_\alpha(in, \ell_I)\}$
- $\$\langle \ell\ dec \rangle = \$_\alpha(dec, \ell)$

As mentioned above, the two partners independently perform some low-level operations locally to their nodes, labelled ℓ_O and ℓ_I. Each label leads to a delay in the rate of the corresponding action. Thus, the cost of the slower partner corresponds to the minimum cost of the operations performed by the participants in isolation. Indeed the lower the cost, i.e. the rate, the greater the time needed to complete an action and hence the slower the speed of the transition (and the slower $F(t) = 1 - e^{-rt}$ approaches its asymptotic value).

Note that we do not fix the actual cost function: we only propose for it a set of parameters to reflect some features of an idealised architecture. Although very abstract, this suffices to make our point. A precise instantiation comes with the refinement steps from specification to implementations as soon as actual parameters become available.

Example (cont'd) We now associate a rate to each transition in the transition system of the system of nodes N, just N for brevity. To illustrate our methodology, we assume that the coefficients due to the nodes amount to 1. We instantiate the cost functions given above, by using the following parameters in the denominator: (*i*) `e` and `d` for encrypting and for decrypting; (*ii*) `s` and `r` for sending and for receiving; (*iii*) `m` for pattern matching; and (*iv*) `f` for the application of the aggregate function f, whose cost is proportional to the number of its arguments.

- $f_E(a) = 1$
- $f_E(\{E_1, \ldots, E_k\}_{K_0}) = \frac{\mathtt{e}}{\mathtt{s}} \cdot \sum_{i=1}^{k} f_E(E_i) + 1$
- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = \frac{1}{\mathtt{s} \cdot \sum_{i=1}^{i} f_E(E_i)}$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = \frac{1}{\mathtt{r} \cdot k + \mathtt{m} \cdot j}$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{K_0}) = \frac{1}{\mathtt{d} \cdot k + \mathtt{m} \cdot j}$
- $\$_\alpha(f(E_1, \cdots, E_k)) = \frac{1}{\mathtt{f} \cdot k}$

These parameters represent the time spent performing the corresponding action on a single term. Intuitively, the greater the time duration is, the smaller the rate. Since transmission is usually more time-consuming than reception, the rate of a communication is that of output. The rates of the transitions of N and $\hat{N}$ are $c_j = \$(\theta_j)$ and $\hat{c}_j = \$(\hat{\theta}_j)$, and $c_{ji} = \$(\theta_{ji})$ and $\hat{c}_{ji} = \$(\hat{\theta}_{ji})$ $(j \in [4,5],\ i \in [0,1])$.

$$c_0 = c_2 = \tfrac{1}{2\mathtt{s}},\quad c_1 = c_3 = \tfrac{1}{2\mathtt{e}+\mathtt{s}},\quad c_{4i} = \tfrac{1}{8\mathtt{f}+2\mathtt{s}}\quad c_{5i} = \tfrac{1}{\mathtt{s}} \qquad \hat{c}_0 = \hat{c}_2 = \hat{c}_3 = \tfrac{1}{2\mathtt{s}},\quad \hat{c}_1 = \tfrac{1}{2\mathtt{e}+\mathtt{s}},\quad \hat{c}_{4i} = \tfrac{1}{6\mathtt{f}+2\mathtt{s}}\quad \hat{c}_{5i} = \tfrac{1}{\mathtt{s}}$$

For instance, the rate of the second transition is: $c_1 = \$(\theta_1) = \frac{1}{2\mathtt{e}+\mathtt{s}}$, where $\frac{1}{2\mathtt{e}+\mathtt{s}} = min\{\frac{1}{2\mathtt{e}+\mathtt{s}}, \frac{1}{2\mathtt{d}+\mathtt{r}+\mathtt{m}}\}$. Note that our costs can be further refined; we could e.g. make the transmission rate also depend on the distance between the nodes, when non internal to a node.

Stochastic Analysis Now, we transform the transition system N into its corresponding $CTMC(N)$, by using the above rates. Afterwards, we can calculate the actual performance measures, e.g. the throughput or utilisation of a certain resource (see [16] for more details on the theory of stochastic processes).

Actually, the *transition rate* $q(N_i, N_j)$ at which a system jumps from N_i to N_j is the sum of the single rates ϑ_k of all the possible transitions from N_i to N_j. Given a transition system N, the corresponding CTMC has a state for each node in N, and the arcs between states are obtained by coalescing all the arcs with the same source and target in N. Recall that a CTMC can be seen as a directed graph and that its matrix $\mathbf{Q}$, the *generator matrix*, (apart from its diagonal) represents its adjacency matrix. Note that $q(N_i, N_j)$ coincides with the off-diagonal element q_{ij} of the generator matrix $\mathbf{Q}$. Hence, hereafter we will use indistinguishably CTMC and its corresponding $\mathbf{Q}$ to denote a Markov chain. More formally, the entries of $\mathbf{Q}$ are defined as follows.

$$q_{ij} = \begin{cases} q(N_i, N_j) = \sum\limits_{\theta_k} \$(\theta_k) \text{ if } i \neq j \ \wedge\ N_i \xrightarrow{\theta_k} N_j \\ -\sum\limits_{j=0, j\neq i}^{n} q_{ij} \qquad\qquad \text{if } i = j \end{cases}$$

Performance measures are usually obtained by computing the stationary distributions of $CTMCs$, since they should be taken over long periods of time to be significant. The *stationary probability distribution* of a CTMC is $\Pi = (X_0, \ldots, X_{n-1})$ s.t. $\Pi^T \mathbf{Q} = \mathbf{0}$ and $\sum_{i=0}^{n} X_i = 1$, which *uniquely* exists if the transition system is finite and has a cyclic initial state.

Example (cont'd) Consider the transition system corresponding to N that is, as required above, finite and with a cyclic initial state. We derive the following generator matrix $\mathbf{Q_1}$ of $CTMC(N)$ and the corresponding stationary distribution Π_1, where $C = 4\mathtt{s} + 2\mathtt{e} + 2\mathtt{f}$, by solving the system of linear equations $\Pi_1^T \mathbf{Q}_1 = \mathbf{0}$ and $\sum_{i=0}^{n} X_i = 1$, where $\Pi_1 = (X_0, \cdots, X_6)$. Similarly, we can derive the generator matrix $\hat{Q}_1'$ and the corresponding stationary distribution $\hat{\Pi}_1$ for the transition system corresponding to $\hat{N}$, where $\hat{C} = 9\mathtt{s} + 2\mathtt{e} + 3\mathtt{f}$.

$$\mathbf{Q_1} = \begin{bmatrix} -c_0 & c_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -c_1 & c_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -c_2 & c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -c_3 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -(c_{40} + c_{41}) & c_{40} & c_{41} \\ c_{50} & 0 & 0 & 0 & 0 & -c_{50} & 0 \\ c_{51} & 0 & 0 & 0 & 0 & 0 & -c_{51} \end{bmatrix}$$

$$\Pi_1 = \left[\tfrac{\mathtt{s}}{\mathtt{C}},\ \tfrac{(2\mathtt{e}+\mathtt{s})}{2\mathtt{C}},\ \tfrac{\mathtt{s}}{2\mathtt{C}},\ \tfrac{(2\mathtt{e}+\mathtt{s})}{\mathtt{C}},\ \tfrac{(4\mathtt{f}+\mathtt{s})}{2\mathtt{C}},\ \tfrac{\mathtt{s}}{4\mathtt{C}},\ \tfrac{\mathtt{s}}{4\mathtt{C}} \right]$$

$$\hat{\Pi}_1 = \left[\tfrac{2\mathtt{s}}{\hat{\mathtt{C}}},\ \tfrac{(2\mathtt{e}+\mathtt{s})}{\hat{\mathtt{C}}},\ \tfrac{2\mathtt{s}}{\hat{\mathtt{C}}},\ \tfrac{2\mathtt{s}}{\hat{\mathtt{C}}},\ \tfrac{(3\mathtt{f}+\mathtt{s})}{\hat{\mathtt{C}}},\ \tfrac{\mathtt{s}}{2\hat{\mathtt{C}}},\ \tfrac{\mathtt{s}}{2\hat{\mathtt{C}}} \right]$$

To define performance measures for a system N, we define the corresponding *reward structure*, following [9,8]. Usually, a reward structure is a function that associates a reward with any state passed through in a computation of N. We compute rewards from rates of transitions [17]. To measure the throughput of a system, i.e. the amount of useful work accomplished per unit time, a reasonable choice is to use as nonzero reward a value equal to the rate of the corresponding transition. The reward structure of a system N is a vector of rewards, whose size amounts to the number of N states. By looking at the stationary distribution and varying the reward structure, we can compute different performance measures. The *total reward* is obtained by summing the values of the stationary distribution Π multiplied by the corresponding reward structure ρ.

Definition 2. *Given a system N, let $\Pi = (X_0, \ldots, X_{n-1})$ be its stationary distribution and $\rho = \rho(0), ..., \rho(n-1)$ be its reward structure. The* total reward *of N is computed as $R(N) = \sum_i \rho(i) \cdot X_i$.*

Example (cont'd) To evaluate the relative efficiency of the two systems of nodes, we compare the throughput of both, i.e. the number of instructions executed per time unit. The throughput for a given activity is found by first associating a transition reward equal to the activity rate with each transition. In our systems each transition is fired only once. Also, the graph of the corresponding CTMC is cyclic and all the labels represent different activities. Therefore the throughput of all the activities is the same, and we can freely choose one of them to compute the throughput of N. Thus we associate a transition reward equal to its rate with the last communication and a null transition reward with all the others communications. The total reward $R(N)$ of the system then amounts to $\frac{1}{2(8\mathtt{s}+4\mathtt{e}+4\mathtt{f})}$, while $R(\hat{N})$ to $\frac{1}{2(9\mathtt{s}+2\mathtt{e}+3\mathtt{f})}$. By comparing the two throughputs, it is straightforward to

obtain that $R(N) < R(\hat{N})$, i.e. that, as expected, $\hat{N}$ performs better. To use this measure, it is necessary to instantiate our parameters under various hypotheses, depending on several factors, such as the network load, the packet size, and so on. Furthermore, we need to consider the costs of cryptographic algorithms and how changing their parameters impact on energy consumption and on the guaranteed security level (see e.g. [15]).

4 Conclusions

In the IoT scenario security is critical but it is hard to address in an affordable way due to the limited computational capabilities of smart objects. We have presented the first steps towards a formal framework that supports designers in specifying an IoT system and in estimating the cost of security mechanisms. A key feature of our approach is that quantitative aspects are symbolically represented by parameters. Actual values are obtained as soon as the designer provides some additional information about the hardware and the network architecture and the cryptographic algorithms relative to the system in hand. By abstractly reasoning about these parameters designers can compare different implementations of the same IoT system, and choose the one that ensures the best trade-off between security guarantees and their price. In practice, we considered a subset of the process algebra IoT-LySA [4] and we adapted the technique in [1] to determine the costs of using/not using cryptographic measures in communications and to reason about the cost-security trade-offs. In particular, we defined an enhanced semantics, where each system transition is associated with a rate in the style of [7,17]. From the rates we derive a CTMC, through which we could perform cost evaluation, by using standard techniques and tools [19,22]. As future work, we plan to assess our proposal considering a more complete case study and considering different metrics as time, network bandwidth and energy consumption.

Our approach follows the well-established line of research about performance evaluation through process calculi and probabilistic model checking (see [10,11] for a survey). To the best of our knowledge, the application of formal methods to IoT systems or to wireless or sensor networks have not been largely studied and only a limited number of papers in the literature addressed the problem from a process algebras perspective, e.g. [13,12,14,21], to cite only a few. In [5] the problem of modelling and estimating the communication cost in an IoT scenario is tackled through Stochastic Petri Net. Their approach is similar to ours: they derive a CTMC from a Petri Net describing the system and proceed with the performance evaluation by using standard tools. Differently from us, they focus not on the cost of security but only on the one of communication (they do not use cryptographic primitives). In [20] a performance comparison between the security protocols IPSec and DTLS is presented, in particular by considering their impact on the resources of IoT devices with limited computational capabilities. They modified protocols implementations to make them properly run on the devices.

An extensive experimental evaluation study on these protocols shows that both their implementations ensure a good level of end-to-end security.

References

1. Bodei, C., Buchholtz, M., Curti, M., Degano, P., Nielson, F., Nielson, H.R., Priami, C.: On evaluating the performance of security protocols. In: Proc. of PaCT'05. LNCS, vol. 3606, pp. 1 – 15. Springer (2005)
2. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. Journal of Computer Security 13(3), 347–390 (2005)
3. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: A step towards checking security in IoT. In: Procs. of ICE 2016. EPTCS, vol. 223, pp. 128–142 (2016)
4. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Where do your IoT ingredients come from? In: Procs. of Coordination 2016. LNCS, vol. 9686. Springer (2016)
5. Chen, L., Shi, L., Tan, W.: Modeling and performance evaluation of internet of things based on petri nets and behavior expression. Research Journal of Applied Sciences, Engineering and Technology 4(18), 3381–3385 (2012)
6. Degano, P., Priami, C.: Non interleaving semantics for mobile processes. Theoretical Computer Science 216 (1999)
7. Degano, P., Priami, C.: Enhanced operational semantics. ACM Computing Surveys 33(2), 135 – 176 (July 2001)
8. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
9. Howard., R.: Dynamic Probabilistic Systems: Semi-Markov and Decision Systems, vol. Volume II. Wiley (1971)
10. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Procs. of Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07). vol. LNCS 4486, pp. 220–270 (2007)
11. Kwiatkowska, M., Parker, D.: Advances in probabilistic model checking. In: Procs. of Software Safety and Security - Tools for Analysis and Verification. vol. 33, pp. 126–151. IOS Press (2012)
12. Lanese, I., Bedogni, L., Felice, M.D.: Internet of things: a process calculus approach. In: Procs of Symp. on Applied Computing (SAC '13). pp. 1339–1346. ACM (2013)
13. Lanese, I., Sangiorgi, D.: An operational semantics for a calculus for wireless systems. Theor. Comput. Sci. 411(19), 1928–1948 (2010)
14. Lanotte, R., Merro, M.: A semantic theory of the Internet of Things. In: Procs. of Coordination 2016. LNCS, vol. 9686, pp. 157–174. Springer (2016)
15. Lee, J., Kapitanova, K., Son, S.: The price of security in wireless sensor networks. Computer Networks 54(17), 2967–2978 (2010)
16. Nelson, R.: Probability, Stochastic Processes and Queeing Theory. Springer (1995)
17. Nottegar, C., Priami, C., Degano, P.: Performance evaluation of mobile processes via abstract machines. Transactions on Software Engineering 27(10) (2001)
18. Priami, C.: Language-based performance prediction of distributed and mobile systems. Information and Computation 175, 119–145 (2002)
19. Reibnam, A., Smith, R., Trivedi, K.: Markov and Markov reward model transient analysis: an overview of numerical approaches. European Journal of Operations Research (40), 257–267 (1989)
20. Rubertis, A.D., Mainetti, L., Mighali, V., Patrono, L., Sergi, I., Stefanizzi, M., Pascali, S.: Performance evaluation of end-to-end security protocols in an Internet of Things. In: Proc. of (SoftCOM). pp. 1–6. IEEE (2013)

21. Singh, A., Ramakrishnan, C.R., Smolka, S.: A process calculus for mobile ad hoc networks. Sci. Comput. Program. 75(6), 440–469 (2010)
22. Stewart, W.J.: Introduction to the numerical solutions of Markov chains. Princeton University Press (1994)

A semantics for disciplined concurrency in COP*

Matteo Busi, Pierpaolo Degano, and Letterio Galletta

Dipartimento di Informatica, Università di Pisa
m.busi@studenti.unipi.it, {degano,galletta}@di.unipi.it

Abstract. A concurrent extension of the recent COP language $\mathrm{ML}_{\mathrm{CoDa}}$ is presented. We formalise its operational semantics and we propose a run time verification mechanism that enforces a notion of non-interference among concurrent threads. More precisely, this mechanism prevents an application from modifying the context so as to dispose some resources or to contradict assumptions upon which other applications rely.

1 Introduction

Modern software have to run in highly dynamic and open heterogeneous environment, often referred as the context. The context abstracts the communication infrastructure and the available resources, so as to make them seem less heterogeneous, unlimited and fully dedicated to their users. Programming these systems thus requires new programming language features and effective mechanisms to deal with context-awareness, i.e. sensing the context, reacting and properly adapting the program behaviour to changes of the actual context. Recently the last two authors proposed $\mathrm{ML}_{\mathrm{CoDa}}$ a two-component Context-oriented Programming (COP) language [10]. It has a logical constituent for specifying and manipulating the context and a functional one for computing. Separation of concerns drove the design choices. Indeed, one specifies the context and its evolution, using its own specific mechanisms and rules, that are typically different from those used in programming applications. Designing a context requires skills different from those needed for applications, and it is usually programmed by requirements engineers [19]. The declarative approach allows requirements engineers to express *what* information the context has to include, leaving to the virtual machine *how* this information is actually collected and managed.

In $\mathrm{ML}_{\mathrm{CoDa}}$ a context is a Datalog knowledge base [15]. Thus, verifying whether a given property holds in the context simply consists in querying a Datalog goal. During the needed deductions the relevant information is also retrieved. The

* Partially supported by the Università di Pisa PRA_2016_64 Project *Through the fog.*

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 177–189 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

choice of a functional language was driven by the popularity of this paradigm (see e.g. F#, Scala), by its formal elegance and the conciseness of its programs. The first mechanism takes care of those program variables that assume different values depending on the different properties of the current context. The notion of *context-dependent binding* makes that explicit. The second one extends standard *behavioural variations*, that are chunks of code that are dynamically activated depending on the context, so adapting the behaviour of the program. In $\mathrm{ML}_{\mathrm{CoDa}}$ behavioural variations are a first-class construct, so they can be referred to by identifiers, and passed to and returned by functions. This helps in programming dynamic adaptation patterns, as well as reusable and modular code. Also, a behavioural variation can be supplied by the context, and then composed with existing ones.

We study these aspects from a basic point of view, in [10] a single application within a context was considered. $\mathrm{ML}_{\mathrm{CoDa}}$ was equipped with an operational semantics which provided us with the basis for a prototypical implementation in F# [5]. Since adaptive applications may misbehave because at design time an unknown environment was not considered, a static analysis ensures that this kind of run time errors never occur, e.g. because the actual hosting environment lacks a required capability. The analysis is performed in two phases: a type and effect system (*at compile time*) and a Control Flow Analysis (*at load time*). Type-checking a program also computes an *effect* that over-approximates the capabilities required by an application at run time. When entering a new context, *before* running the program, this abstraction is exploited to check that the actual context, and those resulting from its evolution, support the capabilities required by the application. Note that this last analysis can only be done at load time, because at compile time the possible hosting contexts are still unknown.

A first extension of $\mathrm{ML}_{\mathrm{CoDa}}$ with concurrency is in [11], where there is a two-threaded system: the context and the application. The first virtualizes the resources and the communication infrastructure, as well as other software components running within it. Consequently, the behaviour of a context, describing in particular how it is updated, abstractly accounts for all the interactions of the entities it hosts. The other thread is the application and the interactions with the other entities therein are rendered as the occurrence of asynchronous events that represent the relevant changes in the context. The semantics of [11] also offered a way of preventing a context change to affect the validity of a choice of a behavioural variation. Also a recovery mechanism is triggered at need.

Here we extend this approach by explicitly describing many applications that execute in a context, and possibly exchange information through it and asynchronously update it. The well known problem of interference now arises, because one thread can update the context possibly making unavailable some resources or contradicting assumptions that another thread relies upon. Classical techniques for controlling this form of misbehaviour, like locks, are not satisfying, because they contrast with the basic assumption of having an open world where applications appear and disappear unpredictably, and freely update the context. However, application designers are only aware of the relevant fragments of the

context and cannot anticipate the effects a change may have. Therefore, the overall consistency of the context cannot be controlled by applications.

The novelty of the semantics proposed here consists in addressing this problem through a run time verification mechanism. We assume our applications be typed as in [10], and the resulting effect, called *history expression* is carried on by the code. Roughly, a history expression collects the sequence of context modifications that an application may perform, as well as the Datalog goals it queries. Intuitively, the effects of the running applications are checked to make sure that the execution of the selected behavioural variation will lead no other application to an inconsistent state, e.g. by disposing a shared resource. Dually, also the other threads are checked to verify that they cause no harm to the application entering in a behavioural variation. Differently than in [10], the verification is not done at load time, but it occurs at run time when a behavioural variation is about to be evaluated. All the checks sketched above are performed on the effects computed at compile time using the Control Flow Analysis of [10]. Note that performing the verification mechanism at load time will result is a huge loss of precision of the analysis due to the inherent non-determinism. At the moment, we designed no recovery mechanisms for when a possible inconsistency is predicted, and we only leave stuck the application responsible for that.

Structure of the paper The next section intuitively presents our COP language and the verification mechanism through an example. In Section 3 we formally define the syntax and the operational semantics of this extension of $\mathrm{ML}_{\mathrm{CoDa}}$. The last section concludes and discusses some related work.

2 An example: competing for visors

Here we elaborate on the example of [10] describing a museum guide. First, we briefly recall the features of the functional component of $\mathrm{ML}_{\mathrm{CoDa}}$, omitting the Datalog constituent that is fully standard. We refer the reader to [10] for a full description of the language.

The original functional component of $\mathrm{ML}_{\mathrm{CoDa}}$ provides two main mechanisms for adaptation. The first is *context-dependent binding* through which a programmer declares variables whose values depend on the context. The construct `dlet x = e1 when G in e2` means that the variable `x` (called *parameter* hereafter) may denote different objects, with different behavior depending on the different properties of the current context, checked by evaluating the goal `G`. If the goal `G` is true in the current context, the variable `x` is bound the result of evaluation of the expression `e1`.

The second mechanism is based on the notion of *behavioural variations*. Basically, it is a list of pairs `(x){G1.e1, ...,Gn.en}`, similar to a case statement, that alters the control flow of applications according to which goal holds in the context, so as to dynamically adapt the running application. Behavioural variations are similar to functions: they take arguments (e.g. `x`) and are (high-order) values so facilitating programming dynamic adaptation patterns. To run a behavioural

variation we need to apply it through the application operator `#(bv, v)` where `bv = (x){G1.e1, ...,Gn.en}`. The application triggers a *dispatching mechanism* that visits the cases in textual order and selects the first expression `ei` whose goal `Gi` holds; then `ei` evaluates in a environment with a new binding between `x` and `v`. If no goal holds then the application cannot adapt to the context and a run time error occurs. The interaction with the Datalog context is not limited to queries, but one can change the knowledge base through `tell`/`retract` operations that add/remove facts.

We illustrate now how the linguistic extensions to $\mathrm{ML}_{\mathrm{CoDa}}$ we are proposing help in designing an adaptive museum guide application. To make our point, it suffices to consider two concurrent applications that are hosted in the shared context offered by the museum intranet.

Each visitor registers at entrance and gets credentials to access the museum intranet and to download the guide application to his smartphone. This guide adapts to the device (e.g. enabling/disabling particular features like HD videos or NFC communication) and to the user's preferences (e.g. accessibility options for blind or deaf people) and has the ability to interact with (some of) those exhibits of the museum which are interactive. Differently than in [10] we here explicitly consider applications that are deployed at an interactive exhibit and reply to user's questions, e.g. about the author of the exhibit.

Since the museum resources can typically be concurrently accessed by a limited number of users, the activities performed by the guide applications and those done by the context itself have to be coordinated.

Here we focus on the operations performed by the guide applications to access the shared interactive exhibits. We assume that applications communicate with a central server; and as in [11] that the shared context provides applications with a communication infrastructure accessible through the `tell`/`retract` operations that update the context, as well as through suitable remote procedure calls (RPCs).

2.1 The context

Abstractly the context could be thought as a heterogeneous collection of data coming from different sources and having different representations. As we said, the context in $\mathrm{ML}_{\mathrm{CoDa}}$ is a knowledge base implemented as a Datalog program, i.e. a set of facts that predicate over a possibly rich data domain, and a set of logical rules to deduce further implicit properties of the context itself. Below, we briefly introduce some aspects of the context of the museum where the multimedia guide is plugged in.

Suppose we have the museum context presented in [10], that includes information about the user profiles, their device capabilities, the ticketing policies, access points to the intranet etc. Here we enrich the museum context with some facts about the exhibits, e.g. the following fact declares that exhibit `x` is interactive:

```
is_interactive(x)
```

A specific exhibit can interact with a visitor through a virtual reality visors that plays a video. This feature can be expressed in the context by the following fact

```
play_video(x,visor) :-
 is_interactive(x), has_visors(x,visor)
```

Acquiring a visor requires to check if it is available, i.e. that the following Datalog goal holds

```
← ¬busy(x, visor)
```

If this is the case, the application can lock the visor by inserting the fact `busy(x, visor)` in the context through the `tell` construct. Symmetrically, releasing the visor is done by removing the fact through a `retract`.

2.2 The guide and the exhibit application

We now show the relevant code concerning the interaction among the multimedia guide, the exhibit application and the shared environment. For readability we use a sugared syntax of our extended $\mathrm{ML}_{\mathrm{CoDa}}$ which will be formally introduced in Section 3. Assume that a new GUI element in the guide is enabled when data are downloaded from the interactive exhibit. Once active it allows a user to visualize the data of the exhibit.

Suppose that a user `U` wants to interact with the virtual reality exhibit `ie` with two visors `v1` and `v2`. As expected `U` can acquire a visor if it is available and cannot if in use until it is released. The following code implements the above:

```
fun interact () =
 let get_visor = (){
  ← ¬busy(ie, v1).
   showMessage "Please use the first visor"
   enable_first()
  ← busy(ie, v1), ¬busy(ie, v2).
   showMessage "Please use the second visor"
   enable_second()
  ← busy(ie, v1), busy(ie, v2).
   showMessage "Please wait..."
 }
 in #(get_visor, ())
```

In the code we define the behavioural variation `get_visor` with no argument that queries the context to get information on availability of visors. The behavioural variation is applied in the last line through the `#` construct. Each case of `get_visor` is driven by a goal, e.g. `¬busy(ie, v1)`. The application interacts with visors via RPCs.

In the exhibit the implementation of the RPC function `enable_first` is straightforward:

```
fun enable_first () =
 tell busy(exhibitID,v1)
 (* Code for interacting with the user *)
 retract busy(exhibitID,v1)
```

where `exhibitID` identifies the current exhibit; the code for the function `enable_second` is analogous.

2.3 Executing the guide

Assume Alice is in front of one of the interactive exhibits and wants to play with it. She taps on the relevant button to launch the function `interact`, causing the behavioural variation `get_visor` to run. If the visor `v1` is available, the first goal succeeds and the RPC `enable_first` is invoked.

Now Bob arrives and wants to interact with the same exhibit. Three different situation may occur, depending on the execution point reached by Alice's application when Bob starts to execute the behavioural variation `get_visor`:

- Alice has still to execute `tell(busy(v1))` and thus also Bob could get the visor `v1`. The runtime of ML_{CoDa} first inspects the history expression H associated with Bob at compile time, and discovers the potential damage to Alice. Indeed H records that Bob will change the context through `tell busy(exhibitID,v1)`, so falsifying the goal ¬`busy(ie, v1)` that Alice has just checked. In this case, the runtime prevents Bob from performing the harmful operation;
- Alice completed the execution of `tell(busy(v1))` and is interacting with the exhibit. Then Bob will simply find the visor `v1` busy and the second case of his behavioural variation will be selected;
- Alice has released the visor `v1` through `retract(busy(v1))` and so Bob can acquire it.

As intuitively described above, the extended runtime support of ML_{CoDa} embeds a verification mechanism at run time, so enforcing a sort of non-interference property among threads. Of course, the simple situation above can be extended to the case with many visitors interacting with the same exhibit.

3 Regulating concurrency in ML_{CoDa}

This section presents our extension of ML_{CoDa} with concurrency. As in [10] the context provides applications with information and resources they need. Here, the context works also as a shared memory through which applications interact. Additionally, our semantics makes sure that when an application modifies the context, it falsifies no hypothesis that drove the selection of the running behavioural variations of other applications.

Syntax The Datalog part is standard: a program is a finite set of (ground) facts and clauses. As defined in [8], we assume that each program is *safe* and *stratified*, so negation is allowed.

The functional part inherits most of the ML constructs. In addition to the usual ones, our values include Datalog facts F and behavioural variations. Moreover, we introduce the set $\tilde{x} \in DynVar$ of *parameters*, i.e., variables assuming

values depending on the properties of the running context; while $x, f \in Var$ are standard identifiers, with the proviso that $Var \cap DynVar = \emptyset$. The syntax of ML_{CoDa} follows:

$$
\begin{aligned}
Va ::= & G^l.e \mid G^l.e, Va \\
v ::= & c \mid \lambda_f x.e \mid (x)\{Va\} \mid F \\
e ::= & v \mid x \mid \tilde{x} \mid e_1\, e_2 \mid \textbf{if}\, e_1\, \textbf{then}\, e_2\, \textbf{else}\, e_3 \mid \textbf{let}\, x\, =\, e_1\, \textbf{in}\, e_2 \mid \\
& \textbf{dlet}\, \tilde{x}\, = e_1\, \textbf{when}\, G^l\, \textbf{in}\, e_2 \mid \textbf{tell}(e_1)^l \mid \textbf{retract}(e_1)^l \mid \#(e_1, e_2) \mid \lfloor e \rfloor_G
\end{aligned}
$$

The novelties w.r.t. [10] are that the goals of behavioural variations $(x)\{Va\}$ and of the context dependent binding **dlet** have labels $l \in Lab$ to link them with their abstract counterparts in history expressions (see below). These labels are mechanically attached (in the abstract syntax tree) and uniquely identify sub-expressions. They do not alter the semantics of [10]: at run time, the first goal G_i^l satisfied by the context determines the expression e_i to be run (*dispatching*). Also the **tell**/**retract** constructs, which insert/remove facts from the context, carry labels. The application of a behavioural variation $\#(e_1, e_2)$ which applies e_1 to its argument e_2 is the same as in [10]: the dispatching mechanism is triggered to query the context and to select from e_1 the expression to run. In the formal development we record the goal selected by the dispatching mechanism through the auxiliary expression $\lfloor e \rfloor_G$.

Semantics We assume that our systems are made of some expressions running concurrently in a context $C \in Context$. Here we inherit the standard top-down semantics [8] for Datalog under the Closed World Assumption to deal with negation. We write $C \vDash G\, with\, \theta$ when the goal G, under a ground substitution θ, is satisfied in the context C. The concurrent semantics of a system is defined by a hierarchy of three SOS transition systems. The first one is for expressions with no free variables, but possibly with free parameters, thus allowing for openness. It is a slight modification of the one in [10], where the environment ρ: $DynVar \to Va$ maps parameters to variations Va. A first novelty is that transitions are labelled to record the actions performed (irrelevant labels will be omitted). For example we have the following axiom that specifies how the fact F is added to the context C. It also records where this happens through the label that identifies the specific **tell** responsible for that. This information will be used later on to link the actual code with its history expression, computed by the type and effect system, and it helps the verification made at run time.

$$
\frac{}{\rho \vdash C, \textbf{tell}(F)^l \xrightarrow{\ l\ } C \cup \{F\}, ()} \quad \textsc{Tell2}
$$

A second novelty concerns the rules that query the context. Through the dispatching mechanism (see below), we detect a case e of a behavioural variation which will be selected and run (suitable instantiated). Also here the transition records the label ℓ of the goal G satisfied, for future use. Additionally, the goal G indexes the selected case, giving raise to the auxiliary expression $\lfloor e \rfloor_G$. Indeed,

G has to always hold along the execution of e, until it reduces to a value v; in other words, $\lfloor e \rfloor_G$ reduces to v.

$$\frac{\rho(\tilde{x}) = Va \qquad dsp(C,\ Va) = (e,\ \{\overrightarrow{c}/\overrightarrow{y}\}, G^l)}{\rho \vdash C,\ \tilde{x} \xrightarrow{l} C,\ \lfloor e\{\overrightarrow{c}/\overrightarrow{y}\}\rfloor_G} \textsc{Dyvar}$$

where dispatching is essentially the same of [10]:

$$dsp(C,\ (G^l.e,\ Va)) = \begin{cases} (e,\ \theta, G^l) & \text{if } C \vDash G\, with\, \theta \\ dsp(C,\ Va) & \text{otherwise} \end{cases}$$

Also the rules for behavioural variation applications are modified similarly. Labels are preserved by the inference rules.

The second level provides the third one with the relevant information to guarantee that no applications modify a resource needed by another one. To do that, we exploit the behavioural abstraction of the application computed by the type and effect system of [10] in order to perform run time checks. We recall from [10] the syntax of the abstractions, called history expressions $H \in \mathbb{H}$, that here carry labels $\ell \in \widehat{Lab}$, for simplicity disjoint from Lab.

$$H ::= \epsilon \mid h \mid \mu h.H \mid tell\, F^\ell \mid retract\, F^\ell \mid H_1 + H_2 \mid H_1 \cdot H_2 \mid \Delta$$
$$\Delta ::= ask\, G^\ell.H \otimes \Delta \mid fail$$

History expressions abstractly represent the activities performed: $tell/retract$ are obvious, $\mu h.H$ is for recursion, $+$ abstracts conditionals, $\cdot$ sequential compositions and Δ represents the dispatching mechanism. As in [4], our type and effect system associates with an expression e a (standard) type, an effect H and a function Λ that records the correspondence of labels ℓ in H with those in e. The semantics of history expression is trivially extended to take care of labels.

There are three rules in the second level. The first follows:

$$\frac{\emptyset \vdash C, e \xrightarrow{l} C', e' \qquad C, H \rightarrow^* C, H'' \xrightarrow{\ell} C', H'}{C,\ e : (H, \omega) \rightarrow C',\ e' : (H', \omega \wedge \widehat{G})}\ \Lambda(\ell) = l$$

where $\widehat{G} = \begin{cases} G & \text{if } ask\, G^\ell.H \text{ is a sub-history of } H \\ true & \text{otherwise} \end{cases}$

We write $e : (H, \omega)$, when H is the abstraction of e and ω is the conjunction of all the goals (holding in C) of the behavioural variations still in execution. The case $\widehat{G} = true$ holds when l labels a $tell$ or a $retract$.

The second rule governs the termination of a behavioural variation and the elimination of the relevant goal:

$$\frac{\emptyset \vdash C, \lfloor v \rfloor_G \rightarrow C, v}{C,\ e : (H, \omega \wedge G) \rightarrow C,\ v : (H, \omega)}$$

The third rule considers the case when the context does not change (we do not track the changes in H):

$$\frac{\emptyset \vdash C, e \to C, e'}{C,\ e : (H, \omega) \to C,\ e' : (H, \omega)}$$

The top-level transition system takes care of the (interleaved) concurrent behaviour of systems. Here we assume the standard congruences of $\|$, the parallel operator, e.g. commutativity. The first rule of this level is

$$\frac{C, e_0 : (H_0, \omega_0) \to C, \lfloor e_0' \rfloor_G : (H_0', \omega_0 \wedge G) \qquad \alpha_1 \qquad \alpha_2}{\begin{array}{c} C,\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ e_0 : (H_0, \omega_0) \to \\ C,\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ \lfloor e_0' \rfloor_G : (H_0', \omega_0 \wedge G) \end{array}}$$

where α_1 and α_2 are the following conditions:

$$\alpha_1 = \forall C'' s.t.\ C, H_0 \to^* C'', H_0''.\ C'' \models \bigwedge_{i=1}^{n} \omega_i$$

$$\alpha_2 = \forall i \in [1, n] \forall C'' s.t.\ C, H_i \to^* C'', H_i''.\ C'' \models \omega_0 \wedge G$$

The first condition says that no actions of e_0 will falsify any of the goals of the e_i. Symmetrically, the second one guarantees that the goals of e_0 will hold along the execution of the other threads.

There is a rule for when the context changes because of a *tell*/*retract*:

$$\frac{C, e_0 : (H_0, \omega_0) \to C', e_0' : (H_0', \omega_0) \qquad \alpha_1 \qquad \alpha_2}{C,\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ e_0 : (H_0, \omega_0) \to C',\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ e_0' : (H_0', \omega_0)}\ C \neq C'$$

The last rule is for when the context does not change, and no violations may then occur

$$\frac{C, e_0 : (H_0, \omega_0) \to C, e_0' : (H_0, \omega_0)}{C,\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ e_0 : (H_0, \omega_0) \to \qquad C,\ \|_{i=1}^n\ e_i : (H_i, \omega_i)\ \|\ e_0' : (H_0, \omega_0)}$$

The mechanism specified by the last two inference rules prevents all the applications running concurrently to misbehave so causing adaptivity errors each other. The properties of the context and the resources acquired by an application in order to execute a behavioural variation are guaranteed to hold until the behavioural variation itself is not completed, regardless of any update made by other applications. The conditions α_1 and α_2 are crucial for performing these checks at run time. These conditions can be verified through the control flow analysis of [10]. Essentially, exploiting the history expressions H_i and H_0 the analysis results in a graph $\mathcal{G}$ that describes the possible evolutions of the context C. Technically, the graph $\mathcal{G}$ is obtained as solution of a set of constraints following the Flow Logic approach [16]. Very roughly, these constraints express how a *tell* or a *retract* inside a history expression modifies a context into a new one. Note that there this static analysis is done at load time, while here it has to be performed at

run time, because one only knows the acquired resources while executing. Note also that condition α_1 constrains the effects of the running application e_0 on the other applications, but not on itself, otherwise a wanted, and fairly acceptable behaviour could be discarded. An example of this is discussed in Section 2: the RPC function `enable_first` above falsifies the goal $\leftarrow \neg$`busy(ie, v1)` driving the first case of the behavioural variation of the function `interact`. Summing up our concurrent semantics embeds a sort of non-interference mechanism.

4 Conclusions

Our starting point has been the two-component COP language ML_{CoDa} [10], in which the context is a Datalog knowledge base and the application code is ML with specific adaptation constructs; in particular, the dispatching mechanism is driven by Datalog goals. Here, we have extended ML_{CoDa} to operate in a concurrent environment. A major contribution of this paper is the run time verification mechanism embedded in the semantics. It is triggered when a behavioural variation is about to start and it enforces a sort of non-interference among the running applications.

Our proposal relies on a formal operational semantics of the extended language, as well as on a type and effect system that associates each application with a safe abstraction of its run time behaviour, namely a history expression. The verification mechanism uses the history expressions of the application ready to evaluate a behavioural variation and checks that none of its future actions may invalidate the assumptions driving the execution of the other threads. Analogously, the application is protected against actions done by other threads. The verification can be performed by simply moving at run time the Control Flow Analysis done in [10] at load time. As a matter of fact, our mechanism for non-interference has been inspired by the classical notion of critical section. In our case, the resources to protect are the properties of the context that are relevant for the execution of applications: a behavioural variation plays here a role similar to that of a critical section.

Future work includes extending our prototypical implementation of ML_{CoDa} [5] with concurrency and the run time verification. Since history expressions are over-approximations of the behaviour of applications, in some cases the verification mechanism unnecessarily suspends the execution of a thread, e.g. when there is a conditional and only one branch may lead to troubles. We would like to investigate whether it will be possible to live dangerously in a partially inconsistent context. However, in this case some notions of compensation (or recovery like in [11]) would be in order to prevent an application to crash definitely. A different approach would be analysing a history expression to detect where the code may perform dangerous activities, and dynamically instrument it accordingly, as proposed in [4].

Related Work Most research efforts in COP have been directed toward the design and the implementation of concrete languages; see [2] for an analysis of some

implementations. Below, we focus on papers that are strictly related to our proposal and on those proposing concurrency and verification mechanism; see [17] for a broad survey on primitives and possible language designs.

Many COP languages are object oriented, thus behavioural variations are often implemented as partially defined methods, and are not values as it is the case in $\mathrm{ML}_{\mathrm{CoDa}}$. The most notable exception is *ContextL* [9], that is based on *Common Lisp*, from which it inherits higher-order features.

Typically, the context is a stack of layers, that can be activated and deactivated at run time. One can simply and intuitively view a layer as an elementary property (a proposition) of the current context. $\mathrm{ML}_{\mathrm{CoDa}}$ differs from this approach having distinct formalisms for specifying the context and the applications. Others papers in the literature do the same. The language *Javanese* [14] supplies primitives for declaring a context and its properties in a logical manner through a temporal logic. In *Javanese* the context represents properties of the system that are "activated by an action and held active until another action that deactivates it occurs". This is similar to our vision where the system running an application is part of the context and where a fact inserted into the context holds until explicitly retracted. Also *Subjective-C* [13] is equipped with a domain-specific language for specifying the contents of what is called a *set of contexts*. A context of *Subjective-C* is just a single property holding in the working environment of an application, behaving much like our facts. Similarly, a context is activated when particular circumstances occur in the environment. Furthermore, *Subjective-C* proposes constructs for specifying relationships and constraints over contexts, e.g. inclusion and conflict. This approach is very similar to ours, and Datalog can also express these kinds of relations through logical rules.

As far as we know a limited number of papers have considered concurrency in COP. In [12] *ContextML*, a predecessor of $\mathrm{ML}_{\mathrm{CoDa}}$, is proposed. It extends ML with layers, layered expressions, and scoped activation mechanisms for layers (`with` and `without`). Applications are made of many components with a local context interacting through message passing. Similarly to the present proposal a type and effect system computes application abstractions, but there they are statically model-checked to enforce communication compliance and security policies. In [18] the formal semantics of *ContextErlang* describes the behaviour of the constructs for adaptation within a distributed and concurrent framework, based on message passing. Similar to ours, that semantics ensures a non-interference property among *Erlang* actors.

As regards event-driven adaptation, *EventCJ* [1] is a Java-based language which combines mechanisms from COP with event based changes of the context. It provides constructs to declare both the events thrown by an application and the transition rules specifying how to change the context when an event is received. The language *Flute* [3] is designed for programming reactive adaptive software. *Flute* constrains the execution of a procedure with certain contextual properties specified by a developer. If any of these properties is no longer satisfied, the execution is suspended until the property holds again. Stopping the execution

is like in our approach when goal of a behavioural variation has been falsified because of a context change.

In [6] a run time verification mechanism based on symbolic execution is proposed. Differently from ours the verification step is performed just before activating/deactivating a layer in the context, in order to check whether adaptation is possible. A different approach to verify applications is proposed in [7]: the structure of contexts is a(n enriched) Petri net which is analysed through existing tools.

References

1. Aotani, T., Kamina, T., Masuhara, H.: Featherweight EventCJ: a core calculus for a context-oriented language with event-based per-instance layer transition. pp. 1:1–1:7. COP '11, ACM, New York, NY, USA (2011)
2. Appeltauer, M., Hirschfeld, R., Haupt, M., Lincke, J., Perscheid, M.: A comparison of context-oriented programming languages. In: COP '09. pp. 6:1–6:6. ACM, New York, USA (2009)
3. Bainomugisha, E.: Reactive method dispatch for Context-Oriented Programming. Ph.D. thesis, Comp. Sci. Dept., Vrije Universiteit Brussel (2012)
4. Bodei, C., Degano, P., Galletta, L., Salvatori, F.: Linguistic Mechanisms for Context-aware Security. In: Ciobanu, G., Méry, D. (eds.) ICTAC 2014. LNCS, vol. 8687. Springer (2014)
5. Canciani, A., Degano, P., Ferrari, G.L., Galletta, L.: A context-oriented extension of F#. In: FOCLASA 2015. EPTCS, vol. 201 (2015)
6. Cardozo, N., Christophe, L., De Roover, C., De Meuter, W.: Run-time validation of behavioral adaptations. In: COP'14s. pp. 5:1–5:6. ACM, New York, NY, USA (2014)
7. Cardozo, N., González, S., Mens, K., Straeten, R.V.D., Vallejos, J., D'Hondt, T.: Semantics for consistent activation in context-oriented systems. Information and Software Technology 58, 71 – 94 (2015)
8. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about Datalog (and never dared to ask). IEEE Trans. on Knowl. and Data Eng. 1(1), 146–166 (Mar 1989)
9. Costanza, P., Hirschfeld, R.: Language Constructs for Context-oriented Programming: An Overview of ContextL. In: DSL '05. pp. 1–10. ACM, New York, NY, USA (2005)
10. Degano, P., Ferrari, G.L., Galletta, L.: A two-component language for adaptation: Design, semantics, and program analysis. IEEE Trans. Software Eng. 42(6), 505–529 (2016)
11. Degano, P., Ferrari, G.L., Galletta, L.: Event-driven adaptation in COP. In: PLACES 2016. EPTCS, vol. to appear
12. Degano, P., Ferrari, G.L., Galletta, L., Mezzetti, G.: Types for coordinating secure behavioural variations. In: Sirjani, M. (ed.) COORDINATION 2012. LNCS, vol. 7274, pp. 261–276. Springer (2012)
13. González, S., Cardozo, N., Mens, K., Cádiz, A., Libbrecht, J.C., Goffaux, J.: Subjective-c. In: Malloy, B., Staab, S., van den Brand, M. (eds.) Software Language Engineering, LNCS, vol. 6563, pp. 246–265. Springer (2011)
14. Kamina, T., Aotani, T., Masuhara, H.: A unified context activation mechanism. In: COP'13. pp. 2:1–2:6. ACM, New York, NY, USA (2013)

15. Loke, S.W.: Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. Knowl. Eng. Rev. 19(3), 213–233 (Sep 2004)
16. Nielson, H.R., Nielson, F.: Flow logic: A multi-paradigmatic approach to static analysis. In: Mogensen, T., Schmidt, D.A., Sudborough, I. (eds.) The Essence of Computation, Lecture Notes in Computer Science, vol. 2566, pp. 223–244. Springer Berlin Heidelberg (2002)
17. Salvaneschi, G., Ghezzi, C., Pradella, M.: An analysis of language-level support for self-adaptive software. ACM Trans. Auton. Adapt. Syst. 8(2), 7:1–7:29 (Jul 2013)
18. Salvaneschi, G., Ghezzi, C., Pradella, M.: Contexterlang: A language for distributed context-aware self-adaptive applications. Sci. Comput. Program. 102, 20–43 (2015)
19. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. 6(1), 1–30 (Jan 1997)

On Exchange-Robust and Subst-Robust Primitive Partial Words

Ananda Chandra Nayak[1], Amit K. Srivastava[2] and Kalpesh Kapoor[1]

[1] Department of Mathematics
Indian Institute of Technology Guwahati, Guwahati, India
[2] Department of Computer Science & Engg.
Indian Institute of Technology Guwahati, Guwahati, India
{n.ananda,amit.srivastava,kalpesh}@iitg.ernet.in

Abstract. A partial word is a word that may have some unknown places known as "holes" and can be replaced by the symbols from the underlying alphabet. A partial word u is said to be primitive if there does not exist a word v such that u is contained in a nontrivial integer power of v. We study the preservation of primitivity in partial words by the effect of some point mutation operations. In this paper, we investigate the effect of exchanging two adjacent symbols and of substituting a symbol by another symbol from the alphabet. We characterize the classes of primitive partial words with one hole which are not exchange robust and not substitute robust. We prove that the language of exchange robust primitive partial words with one hole is not right 1-dense and also prove that the language of primitive partial words with one hole which is substitute robust is closed under conjugacy relation. We show that the language of non-exchange-robust primitive partial words is not context-free over a binary alphabet.

Keywords: Combinatorics on words, primitive word, partial word, exchange-robust partial word, subst-robust partial word

1 Introduction

Let V be a finite alphabet. A word is a sequence of symbols from the alphabet V. A partial word is a word that may have some unknown positions known as "holes" or "do not know" symbols and are represented by $\Diamond$. The holes in the partial words are place holder for the usual symbols in the alphabet. The study of partial words is motivated by its application in molecular biology [1]. For example, the alignment of two DNA sequences to recover as much information as possible can be seen as the construction of two compatible partial words. The combinatorial

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 190–202 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

properties of words play a vital role in the areas including formal languages [18], coding theory [2], string searching algorithms [6], and computational biology [10].

In study of an algebraic structure, it is interest to investigate the operators that preserves the algebraic structure. For example, homomorphisms are well-known structure-preserving transformations in algebra and word combinatorics. A word is said to be primitive if it cannot be represented as a power of a shorter word. The algorithmic, algebraic and applied combinatorial properties of primitive words have been extensively studied; see for example [11,12]. In [17,13], Păun et al. have studied the conditions to preserve primitivity under morphisms. Dassow et al. [7] investigated some operations where they proved that ww' is a primitive word where w is a given word and w' is a modified mirror image of w. This has been extended by Blanchet-Sadri et al. [5] for partial words. In [16], Păun et al. introduced the notion of robustness of primitive words as point mutation operations such as inserting a symbol, deletion of a symbol, substituting a symbol by another symbol in the primitive word which preserves the primitivity. In this paper, we discuss the robustness of primitive partial words with respect to substitution and exchange operations. These classes of primitive partial words are known as subst-robust and exchange-robust primitive partial words.

The rest of the paper is organized as follows. In Section 2 we discuss the basic concepts and related results that are required in later sections. We characterize the exchange-robust primitive partial words and identify some important properties in Section 3. We prove that the language of non-exchange-robust primitive partial words is not context-free over a given alphabet in Section 4. In Section 5, we characterize the primitive partial words which remain primitive on substituting a symbol by another symbol and identify some important results. Section 6 presents about the conclusion and future work.

2 Prerequisites

Let V be a finite and nontrivial alphabet. A sequence of symbols from V is called a word or string. A total word $w = a_0a_1 \dots a_{n-1}$ of length n is a total function $w : \{0,\ 1, \dots, n-1\} \mapsto V$ where $a_i \in V$ for $i = 0,\ 1, \dots, n-1$. The length of a word w is denoted by $|w|$ and defined as the number of symbols appearing in w. The empty word, λ, does not contain any symbol and $|\lambda| = 0$. The set of all strings over the alphabet V is denoted as V^*. The notation $\alpha(w)$ is used for the set of distinct symbols appearing in w and $|\alpha(w)|$ is the number of distinct symbols in w. A positive integer p is said to be a period of w if $a_i = a_{i+p}$ for $0 \leq i \leq n-p-1$. Let $w = xyz$ be a word. Then y is called a factor of w and x and z are called prefix and suffix of w, respectively. The reverse of a word w is written as $rev(w)$ and defined as $rev(w) = a_{n-1} \cdots a_1a_0$ when $w = a_0a_1 \cdots a_{n-1}$. A word w is primitive if there does not exist a word u such that $w = u^n$ with $n \geq 2$. The language of primitive and nonprimitive words are denoted as Q and Z, respectively [11]. For a language L, we define $length(L) = \{|w| : w \in L\}$.

A partial word u of length n over an alphabet V can be defined by a partial function $u : \{0, \dots, n-1\} \mapsto V$ [4]. A partial word u may contain some

'*do not know*' symbols known as holes along with the symbols from the underlying alphabet. A "hole" or "do not know" symbol is represented by $\Diamond$. For $0 \leq i < n$, if $u(i)$ is defined then we say that $i \in D(u)$ (the domain of u), otherwise $i \in H(u)$ (the set of holes) [3]. A total word is a partial word with empty set of holes. For example, $u = \Diamond bac\Diamond a$ is a partial word of length 6 and $D(u) = \{1, 2, 3, 5\}$ and $H(u) = \{0, 4\}$. We use $V_\Diamond$ to denote enlarged alphabet $V \cup \{\Diamond\}$. If u is a partial word of length n over V, then the companion of u is the total function $u_\Diamond : \{0, \ldots, n-1\} \to V \cup \{\Diamond\}$.

Let u and v be two partial words of equal length. Then u is said to be *contained* in v, if all elements in $D(u)$ are also in the set $D(v)$ and $u(i) = v(i)$ for all $i \in D(u)$ and denoted by $u \sqsubset v$. A partial word u is said to be *compatible* to a partial word v if there exists a partial word w such that $u \sqsubset w$ and $v \sqsubset w$ and is denoted by $u \uparrow v$. A partial word u is said to be primitive if there does not exist a word v such that $u \sqsubset v^n$ with $n \geq 2$ [4]. For example, $w = abb\Diamond$ is a primitive partial word and $u = ab\Diamond b$ is a nonprimitive partial word over $V = \{a,\ b\}$. The languages of primitive partial words and nonprimitive partial words are denoted as Q_p and Z_p, respectively. In particular, Q_p^i denotes the language of primitive partial words with at most i holes. A local period of a partial word u is a positive integer p such that $u(i) = u(i+p)$ whenever $i,\ i+p \in D(u)$ [4].

Let $w = uv$ be a nonempty partial word. Then, the partial words u and v are said to be prefix and suffix of w, respectively. A partial word y is said to be a factor of a word w if w can be written as xyz, where $x,\ z \in V_\Diamond^*$ and $y \in V_\Diamond^+$. The set V_i^* contains all partial words with exactly i-holes. The partial word y is said to be proper factor if $x \neq \lambda$ or $z \neq \lambda$. A prefix (suffix) of length k of a partial word w is denoted as $\text{pref}(w, k)$ ($\text{suff}(w, k)$), respectively, where $k \in \{0, 1, \ldots, |w|\}$ and $\text{pref}(w, 0) = \text{suff}(w, 0) = \lambda$.

We now recall some results from the literature that will be useful later in this paper.

Theorem 1 (Fine and Wilf's Theorem [9]). *Let u and v be nonempty words over V. Suppose u^h and v^k, for some h and k, have a common prefix (or suffix) of length $|u| + |v| - gcd(|u|, |v|)$. Then there exists $z \in V^*$ of length $gcd(|u|, |v|)$ such that $u, v \in z^*$.*

Berstel and Boasson revisited Fine and Wilf's theorem for partial words with one hole.

Theorem 2 (Berstel and Boasson [1]). *Let w be a partial word with one hole which is locally p-periodic and locally q-periodic. If $|w| \geq p + q$ then w is $gcd(p, q)$-periodic.*

Definition 3 (Reflective Language [19]). *A language L is called reflective if $xy \in L$ implies $yx \in L$ for all $x,\ y \in V^*$.*

Lemma 4. *[3] Let u and v be partial words. If there exists a primitive word x such that $uv \sqsubset x^n$ for some positive integer n, then there exists a primitive word y such that $vu \sqsubset y^n$. Moreover, if uv is primitive then vu is primitive.*

Corollary 5. *The languages Q_p and Z_p are reflective.*

Proposition 6. *[3] Let w be a partial word with one hole such that $|\alpha(w)| \geq 2$. If a is any letter, then either w or wa is primitive.*

The following result shows that we can obtain at least one primitive word by substituting a symbol by another symbol.

Proposition 7. *[16] Let $|V| \geq 3$. For any word $x \in V^*$ and for each decomposition $x = x_1 a x_2,\ x_1, x_2 \in V^*,\ a \in V$, there is $b \in V$, $b \neq a$ such that $x_1 b x_2$ is primitive.*

But the result is not true in case of partial words. For example, consider $w = x\Diamond xa$. Substituting a by any letter from V will generate nonprimitive partial words.

Proposition 8. *[3] Let u be a partial word with one hole which is not of the form $x\Diamond x$ where $x \in V^+$. Then for $a \in V$, at most one of the partial words ua is not primitive.*

3 Exchange-Robust Primitive Partial Words with One Hole

In this section, we consider a new formal language known as exchange-robust primitive partial words with one hole which remains primitive when any two consecutive symbols in a partial word are exchanged. In particular, we consider $a \neq \Diamond$ for $a \in V$ because exchanging a and $\Diamond$ will generate different partial words. For example, consider $w = aba\Diamond aa \in Q_p$, but exchanging position 3 and 4 we have $w' = abaa\Diamond a \notin Q_p$.

Definition 9 (Exchange-Robust Partial Words). *A primitive partial word $w = a_0 a_1 \cdots a_{i+1} a_{i+2} \cdots a_{n-1}$ of length n with one hole is said to be exchange-robust if and only if*

$$\text{pref}(w, i)\ .\ a_{i+1} a_i\ .\ \text{suff}(w, n-i-2)$$

is a primitive partial word for all $i \in \{0,\ 1, \ldots n-2\}$.

Remark : If a symbol a and a hole $\Diamond$ are adjacent, we exchange a and $\Diamond$.

We denote Q_p^{1X} as the set of all primitive partial words with one hole which are exchange-robust over the alphabet V. Clearly, the set of all exchange-robust primitive partial words with one hole is a subset Q_p. There are infinitely many primitive partial words with one hole which are exchange-robust. For example, $a^n \Diamond b^n a,\ n \geq 2$ is exchange-robust.

Our next result concerns the exchange of two different symbols at consecutive places in a nonprimitive total word. We prove that the new word we obtained by exchanging any two distinct consecutive symbols at any position in a nonprimitive word results in a primitive word.

Lemma 10. *Let w be a total word with $|\alpha(w)| \geq 2$. If $w = x_1abx_2 \in Z$ with $a \neq b$ then $x_1bax_2 \in Q$.*

Proof. We prove it by contradiction. Let w be a nonprimitive word. Then there exists a unique primitive word u such that $w = u^m,\ m \geq 2$. We can express $w = u^{m_1}u_1abu_2u^{m_2}$ where $u_1abu_2 = u$ and $m_1,\ m_2 \geq 0,\ m_1 + m_2 \geq 1$. Assume for contradiction that $w' = u^{m_1}u_1bau_2u^{m_2} \notin Q$. As we know the languages Q and Z are reflective, then it is enough to consider the word $abu_2u^{m_2}u^{m_1}u_1$. Suppose $abu_2u^{m_2}u^{m_1}u_1 = v^m$ and $bau_2u^{m_2}u^{m_1}u_1 = y^n,\ m,\ n \geq 2$ and $y \in Q$. Let p be the common suffix of v^m and y^n. The words v^m and y^n have common suffix of length $m|v| - 2$ and $n|y| - 2$, respectively. We have $|p| = m|v| - 2 = n|y| - 2$. It is not possible to have $m = n = 2$ which is not feasible.

So at least one of m and n is strictly greater than 2. Without loss of generality, let us assume that $m \geq 3$ and $n \geq 2$. Now,

$$\begin{aligned} & 2|p| = m|v| + n|y| - 4 \\ \Rightarrow\ & |p| = \tfrac{m}{2}|v| + \tfrac{n}{2}|y| - 2 \\ \Rightarrow\ & |p| \geq |y| + |v| + \tfrac{1}{2}|v| - 2\ (\because m \geq 3 \text{ and } n \geq 2) \end{aligned}$$

Since $|v| \geq 2$, we obtain that $|p| \geq |y| + |v| - 1$. Hence by Theorem 1, v and y are powers of the same primitive word which is a contradiction. Thus $bau_2u^{m_2}u^{m_1}u_1 \in Q$ which implies that $w' = u^{m_1}u_1bau_2u^{m_2} \in Q$. □

The above result does not hold for partial words. Consider the partial word $w = a\Diamond baab \in Z_p$. if we exchange b and a, we have $w' = a\Diamond abab \notin Q_p$.

Next we study the primitive partial words with one hole in which exchange of two distinct consecutive symbols results in a nonprimitive partial word. We call this set of partial words as non-exchange-robust primitive partial words with one hole. We denote the set of non-exchange-robust primitive partial words with one hole over an alphabet V as $Q_p^{1\overline{X}}$. It is easy to see that $Q_p^{1\overline{X}} \cup Q_p^{1X} = Q_p^1$.

Definition 11 (Non-exchange-robust Primitive Partial Words). *A primitive partial word with one hole is said to be non-exchange-robust if and only if exchange of two distinct consecutive symbols results in a nonprimitive partial word.*

We give the structural characterization of non-exchange-robust primitive partial words with one hole.

Theorem 12. *A primitive partial word w with one hole is non-exchange-robust if and only if w is contained in some word of the form $u^{k_1}u_1abu_2u^{k_2},\ a,\ b \in V,\ a \neq b$ where $u_1xyu_2 \sqsubset u_1abu_2$ for $x,\ y \in V_\Diamond$ such that $u_1yxu_2 \sqsubset u_1bau_2 = u^m$ with $m \geq 2$.*

Proof. We prove the necessary and sufficient conditions as follows:
($\Rightarrow$) Let w be a primitive partial word with one hole. Suppose $w = v_1xyv_2 \sqsubset u^{k_1}u_1abu_2u^{k_2}$ where $a \neq b$ such that $v_1 \sqsubset u^{k_1}u_1,\ v_2 \sqsubset u_2u^{k_2},\ xy \sqsubset ab$. If we exchange x and y, we get $w' = v_1yxv_2 \sqsubset u^{k_1}u_1bau_2$ such that $u_1bau_2 = u^m$ for

$m \geq 2$. Hence $w' \sqsubset u^k,\ k \geq 2$ where $k_1 + m + k_2 = k$ and thus w is not an exchange-robust primitive partial word.

($\Leftarrow$) Let $w \in Q_p^1$ which is not an exchange-robust partial word. Then there exists at least one consecutive positions where exchanging them makes the partial word nonprimitive. The partial word w can be written as either v_1abv_2 where $v_1,\ v_2 \in V_\Diamond^*$ or $v_1a\Diamond v_2$ or $v_1\Diamond av_2$ where $v_1,\ v_2 \in V^*$. In first case, as we have exactly one hole, it is exactly in one among v_1 or v_2. Let $w' = v_1bav_2 \in Z_p$ that is $w' = v_1bav_2 \sqsubset u^m$ for $m \geq 2$. Now $v_1 \sqsubset u^iu_1$ and $v_2 \sqsubset u_2u^j$ for $i,\ j \geq 0$. Combining both we have $v_1bav_2 \sqsubset u^iu_1bau_2u^j$ where $u_1bau_2 = u^k$ for $k \geq 2$.

The other two cases can be handled similarly. □

Let us define $Q_p^{1\overline{X}} = Q_p^1 \setminus Q_p^{1X}$ where '\' is the set minus operator. There are primitive partial words of arbitrary length which are non-exchange-robust; for example, $(ab)^n\Diamond a(ab)^n$ for $n \geq 1$. We denote the set of exchange-robust (non-exchange-robust, respectively) primitive partial words with arbitrary number of holes by Q_p^X ($Q_p^{\overline{X}}$, respectively). The set of $Q_p^{1\overline{X}}$ is not closed under the cyclic permutation unlike the language of del-robust primitive partial words with one hole [14]. For example, consider the partial word $abbabb\Diamond ab \in Q_p^{1\overline{X}}$. One of the cyclic permutation of the partial word is $ababbabb\Diamond$, which is exchange-robust.

Definition 13 ([16]). *A language L is said to be dense if for any word y there exist $x,\ z \in V^*$ such that $xyz \in L$. Let k be a positive integer. If for every $u \in V^*$ there exists a word $x \in V^*,\ |x| \leq k$ such that $ux \in L$ then L is said to be right k-dense.*

Next we prove that the language of primitive partial words with at most one hole is dense over the alphabet $V_\Diamond$. We show that the language of primitive partial words with one hole which are exchange-robust is not dense.

Lemma 14. *The language Q_p^1 is dense over alphabet $V_\Diamond$ in V_1^*.*

Proof. Consider a partial word w with at most one hole . Let $|w| = n,\ n \geq 1$. There are two different cases depending upon whether w is a primitive partial word or a nonprimitive partial word.

Case A. Let w is a primitive partial word. By choosing $x = \lambda$ and $y = \lambda$ according to definition we have $xwy \in Q_p^1$.

Case B. Let w be a nonprimitive partial word. Here we consider two subcases depending on whether w is contained in power of a symbol from the alphabet or power of a word having at least two different letters.

Case B.1 Let $w \sqsubset a^n,\ n \geq 2$ for some symbol $a \in V$. It can be easily seen that $wb^n \in Q_p^1$ where a and b are two distinct letters. Here $x = \lambda$ and $y = b^n$ for some $b \neq a$ such that $xwy \in Q_p^1$.

Case B.2 Let $w \sqsubset u^k$, where $|\alpha(u)| \geq 2$ and $k|n$. By choosing $x = \lambda$ and $y = b^n$, we have $xwy \in Q_p^1$.

Hence, for every $w \in V_1^*$, there exist $x,\ y \in V_1^*$ such that $xwy \in Q_p^1$. So Q_p^1 is dense over the alphabet $V_\Diamond$ in V_1^*. □

We prove the following proposition.

Proposition 15. *The language Q_p^{1X} is not right 1-dense.*

Proof. It is sufficient to find one partial word for which we cannot find any word which satisfies the condition. Let $w = x\Diamond x$ be a primitive partial word with one hole where $x \in V^*$. Let us assume that w is not an exchange-robust primitive partial word. Here both wa and wb are not primitive. Hence we cannot find a word z with $|z| \leq 1$ for w such that $wz \in Q_p^{1X}$. Thus Q_p^{1X} is not right 1-dense. □

For the above proposition, such partial word exists. For example, take $w = aaba\Diamond aaba$. We have $w \notin Q_p^{1X}$ and if we concatenate a or b at the right end of w then we obtain a nonprimitive partial word.

4 $Q_p^{\overline{X}}$ is not context-free

In this section we prove that the language of non-exchange-robust primitive partial words is not a context-free language over a given alphabet. In our proof, we use the fact that intersection of a CFL and a regular language is also context-free. We also use the result that the family of context-free languages are closed under generalized sequential machine(gsm) mapping, and for details see [8].

Theorem 16. *The language of non-exchange robust partial words is not context-free over the alphabet $V = \{a,\ b\}$.*

Proof. Consider the regular language $R = ba^+ba^+ba^+ba^+$. Consider the language

$$L = \{ba^{n_1}ba^{n_2}ba^{n_3}ba^{n_4} \mid n_1,\ n_2,\ n_3,\ n_4 \geq 1,\ (|n_1 - n_3| \leq 1,\ |n_2 - n_4| \leq 1, \\ |(n_1 + n_2) - (n_3 + n_4)| = 0 \text{ or } 2) \text{ and } (n_1 \neq n_3 \text{ or } n_2 \neq n_4)\} \quad (1)$$

We claim that $Q_p^{\overline{X}} \cap R = L$.
The inclusion $Q_p^{\overline{X}} \cap R \supseteq L$ is easy to observe. For the converse, let us take a word $w = ba^{n_1}ba^{n_2}ba^{n_3}ba^{n_4} \in Q_p^{\overline{X}} \cap R$. As $w \in Q_p^{\overline{X}}$, then w can be represented as $w = u_1abu_2$ such that $u_1bau_2 \in Z$. We have the following possibilities of exchanging.

Case 1. $aba^{n_1-1}ba^{n_2}ba^{n_3}ba^{n_4}$
Case 2. $ba^{n_1-1}ba^{n_2+1}ba^{n_3}ba^{n_4}$
Case 3. $ba^{n_1+1}ba^{n_2-1}ba^{n_3}ba^{n_4}$
Case 4. $ba^{n_1}ba^{n_2-1}ba^{n_3+1}ba^{n_4}$
Case 5. $ba^{n_1}ba^{n_2+1}ba^{n_3-1}ba^{n_4}$
Case 6. $ba^{n_1}ba^{n_2}ba^{n_3-1}ba^{n_4+1}$
Case 7. $ba^{n_1}ba^{n_2}ba^{n_3+1}ba^{n_4-1}$

It is easy to see that all the above cases are in the language $Q_p^{\overline{X}}$ only if we have

(1) $n_1 \neq n_3$ or $n_2 \neq n_4$ (otherwise $ba^{n_1}ba^{n_2}ba^{n_1}ba^{n_2} \notin Q$)
(2) $|n_1 - n_3| \leq 1,\ |n_2 - n_4| \leq 1,\ |(i+j)-(k+l)| = 0$ or 2 (otherwise the word $w' \in Q_p^X$

Hence the inclusion $Q_p^{\overline{X}} \cap R \subseteq L$.

As we know that a CFL is closed under the gsm mapping then using a sequential transducer (a gsm), the language $Q_p^{\overline{X}} \cap R$ can be translated into a new language

$$L' = \{a^{n_1}b^{n_2}c^{n_3}d^{n_4} \mid n_1,\ n_2,\ n_3,\ n_4 \geq 1,\ |n_1 - n_3| \leq 1,\ |n_2 - n_4| \leq 1, \\ |(n_1+n_2)-(n_3+n_4)| = 0 \text{ or } 2 \text{ and } (n_1 \neq n_3 \text{ or } n_2 \neq n_4)\} \quad (2)$$

Now we prove that L' is not a context-free language. Assume for contradiction that L' is context-free. Suppose there exist a constant $N > 0$ which must exist by Ogden's lemma. As L' satisfies Ogden's lemma (see Appendix), then every $w \in L',\ |w| \geq N$ can be decomposed into $w = uvxyz$ such that the following conditions hold: (i) vxy contains at most N marked symbols, (ii) v and y have at least one marked symbol, (iii) and $uv^ixy^iz \in L'$ for all $i \geq 0$.

Consider a string $w = a^{n_1}b^{n_2}c^{n_3}d^{n_4}$ such that $n_1 = N,\ n_2 = N,\ n_3 = N+1$ and $n_4 = N-1$. As $|n_1 - n_3| \leq 1,\ |n_2 - n_4| \leq 1,\ |(n_1+n_2)-(n_3+n_4)| = 0$ and $n_1 \neq n_3,\ n_2 \neq n_4$ then $w \in L'$. Let us mark all the occurrences of b which are at least N of them. Now we can decompose $w = uvxyz$ in such a way that all the conditions of Ogden's lemma are satisfied.

Clearly, neither v nor y contain two different symbols. There are two cases depending on whether vy contains an occurrence of a or not.

(I) Suppose vy does not contain any occurrence of a. In this case, we have $u = a^Nb^{i_1},\ v = b^{m_1},\ x = b^{m_2},\ y = b^{m_3}$ such that $m_1+m_3 \geq 1, k_1 = m_1+m_2+m_3$ and $z = b^{N-(k_1+i_1)}c^{N+1}d^{N-1}$. For $i = 2$, $uvxyz = a^Nb^{N+(m_1+m_3)}c^{N+1}d^{N-1} = a^{p_1}b^{p_2}c^{p_3}d^{p_4}$ which is a contradiction as $|p_2 - p_4| \geq 2$.

(II) Suppose vy contains occurrences of a. Let $v = a^j$ and $y = b^k$ for $j,\ k \geq 1$. If $j < k$, then for a large value of i, we can have $w' = uv^ixy^iz = a^{p_1}b^{p_2}c^{p_3}d^{p_4}$ such that $|p_1 - p_3| > 1$ which is a contradiction. Therefore we must have $j \geq k$. Consider the word uv^ixy^iz which becomes $a^{N-j+ji}b^{N-k+ki}c^{N+1}d^{N-1}$. For $i = 5$, we have $w'' = a^{N+4j}b^{N+4k}c^{N+1}d^{N-1}$ where $|(N+4j)-(N+1)| = 4j-1 \geq 3,\ |(N+4k)-(N-1)| = 4k+1 \geq 5$ and $|(N+4j+N+4k)-(N+1+N-1)| = 4(j+k) \geq 8$ which is a contradiction.

Hence L' is not context-free. As we know that the family of context-free languages is closed under sequential transducers and intersection with regular languages, we conclude that $Q_p^{\overline{X}}$ is also not context-free. □

5 Subst-Robust Primitive Partial Words

In this section we study the set of primitive partial words that remains primitive on substitution of a symbol by another symbol. We refer to the definition of

substitute robust total words [16] and define symbol substitution in partial words as follows. Consider a partial word $x \in V_1^+$. We define $one(x) = \{x_1 b x_2 \mid x = x_1 a x_2,\ x_1,\ x_2 \in V_1^*,\ a, b\ \in V,\ a \neq b\}$. Let $L \subseteq V_\Diamond^*$ and $x \in L$. Then x is called subst-robust (w.r.to L) if $one(x) \subseteq L$.

Definition 17 (Subst-Robust Primitive Partial Words). *A primitive partial word w with one hole is said to be subst-robust if and only if $one(w) \subseteq Q_p^1$.*

Remark: Since $\Diamond$ is considered as a place holder for any of the symbol from the given alphabet, only a symbol $a \in V$ can be substituted by another symbol $b \in V$ such that $a \neq b$.

Proposition 18 ([16]). *If L consists of only subst-robust words, then $L = \{w \in V^* \mid |w| \in length(L)\}$.*

The above proposition is not true in case of partial words with one hole. For example, let $L' = \{a\Diamond b, b\Diamond b, b\Diamond a, a\Diamond a\}$. Though L' is subst-robust, it does not contain all the partial words with one hole of length 3. Next we extend the result of Păun et al. [16] in case of partial words.

Lemma 19. *Let $x = x_1\alpha\beta x_2$ be a partial word with one hole where $\alpha, \beta \in V$ and $|x| \geq 4$. Then at least one of the partial words $x_1\alpha'\beta x_2$, $x_1\alpha\beta' x_2$ is primitive where $\alpha \neq \alpha'$ and $\beta \neq \beta'$.*

Proof. We prove it by contradiction. Consider a partial word x with one hole and $|x| \geq 4$. As $|x| \geq 4$, then x can be written as $x = x_1\alpha\beta x_2$ where $\alpha,\ \beta \in V$, $|x_1 x_2| \geq 2$ and either x_1 or x_2 contains a hole. As Q_p^1 is reflective, then to prove the lemma it is sufficient to prove that at least one of the partial word $x_2 x_1 \alpha'\beta$ or $x_2 x_1 \alpha\beta'$ is primitive.

Assume the contrary. Let $x_2 x_1 \alpha'\beta \sqsubset u^m$ and $x_2 x_1 \alpha\beta' \sqsubset v^n$ for $m,\ n \geq 2$ and $u,\ v \in Q$. It is not possible to have $m = n = 2$ otherwise $u = v$ which is a contradiction. So at least one of $m,\ n$ is greater than 2. without loss of generality, let us assume that $m \geq 3,\ n \geq 2$. Similarly, we cannot have $|u| = 1$; otherwise $x_2 x_1 \alpha'\beta \sqsubset u^m$ implies that $u \in \{a,\ b\}$. Since $\alpha \neq \alpha',\ \beta \neq \beta'$ then $x_2 x_1 \alpha\beta'$ is primitive which is a contradiction to the assumption. Hence we have $|u| \geq 2$.

Now, we have $|x_2 x_1| = m|u| - 2$ and $|x_2 x_1| = n|v| - 2$ which implies that

$$\begin{aligned} 2|x_2 x_1| &= m|u| + n|v| - 4 \\ \Rightarrow |x_2 x_1| &= \tfrac{m}{2}|u| + \tfrac{n}{2}|v| - 2 \end{aligned}$$

As $m \geq 3$ and $n \geq 2$, we can write that $|x_2 x_1| \geq |u| + |v| + \frac{1}{2}|u| - 2$. Also $|u| \geq 2$ implies that $|x_2 x_1| \geq |u| + |v| - 1$. We consider the following cases.
(a) If $|x_2 x_1| = |u| + |v| - 1$ then $m = n = 2$ which leads to a contradiction.
(b) If $|x_2 x_1| > |u| + |v| - 1$ then by Theorem 2, there exist a word y such that $u = y^k$ and $v = y^l$ for some integers k and l. Hence $x_2 x_1 \alpha'\beta \sqsubset y^{km}$ and $x_2 x_1 \alpha\beta' \sqsubset y^{ln}$ which is a contradiction. Thus at least one of the $x_2 x_1 \alpha'\beta,\ x_2 x_1 \alpha\beta'$ is a primitive partial word. □

In the above lemma, $|x| \geq 4$ is necessary. For example, let $x = \Diamond ab$ and substituting a by b or b by a will generate nonprimitive partial words. Lemma 19 does not hold for partial words with at least two holes. For example, $w = \Diamond aa\Diamond$ over $V = \{a,\ b\}$. Substituting first occurrence of a by b or last occurrence of a by b will generate nonprimitive partial words.

We denote Q_p^{1S} as the set of primitive partial words with one hole which remains primitive on substitution of a symbol by another symbol from the given alphabet. There are infinitely many primitive partial words with one hole which are subst-robust. For example, $w = (ab)^n\Diamond$ for $n \geq 2$ is subst-robust. It is worth mentioning here that there are primitive partial words with one hole which are at the same time exchange-robust and subst-robust. An example of such partial word is

$$w_m = \Diamond aba^2b^2 \dots a^m b^m$$

for $m \geq 2$ over $V = \{a,\ b\}$.

Let $w = ab\Diamond a$ be a primitive partial word over $V = \{a,\ b\}$. Substituting last occurence of a by b will generate a nonprimitive partial word. We call the set of primitive partial words as non-subst-robust primitive partial words with one hole which on substitution of a symbol by another symbol results in a nonprimitive partial word and denote by $Q_p^{1\overline{S}}$.

$$Q_p^{1\overline{S}} = \{w \mid w = u_1au_2 \in Q_p^1 \text{ and } w' = u_1bu_2 \notin Q_p^1\}$$

Observe that $Q_p^{1S} = Q_p^1 \setminus Q_p^{1\overline{S}}$. Next, we characterize the set of primitive partial words with one hole which are not subst-robust.

Theorem 20. *A primitive partial word with one hole $w = xay$ where $x,\ y \in V_\Diamond^*,\ a \in V$ is not subst-robust if and only if it is contained in a word of the form $u^{k_1}u_1au_2u^{k_2}$ where $x \sqsubset u^{k_1}u_1,\ y \sqsubset u_2u^{k_2}$ with $k_1 + k_2 \geq 1$ such that $u_1bu_2 = u$ where $a \neq b$.*

Proof. ($\Rightarrow$): Let us assume that $w = xay$ is a non-subst-robust primitive partial word with one hole. Then there exist a position in w in which a symbol can be substituted by another symbol and makes it nonprimitive. Let $w' = xby$ be the nonprimitive partial word and hence $w' = xby \sqsubset u^m,\ m \geq 2$ for some $u \in Q$. Therefore, we have $x \sqsubset u^{k_1}u_1,\ y \sqsubset u_2u^{k_2}$ such that $u_1bu_2 = u$ for $b \neq a$. Hence $w \sqsubset u^{k_1}u_1au_2u^{k_2}$.

($\Leftarrow$): Let w be a primitive partial word with one hole and $w = xay \sqsubset u^{k_1}u_1au_2u^{k_2}$, $k_1 + k_2 \geq 1$ where $x,\ y \in V_\Diamond^*$, $a \in V$ with $x \sqsubset u^{k_1}u_1$ and $y \sqsubset u_2u^{k_2}$. Also it is given that substituting a symbol $b \neq a$, we have $u_1bu_2 = u$. If we substitute a symbol $b \neq a$ in $w = xay$, we get $w' = xby$ such that $w' = xby \sqsubset u^{k_1}uu^{k_2} = u^{k_1+k_2+1}$ and $k_1 + k_2 + 1 \geq 2$. Hence $w = xay$ is not subst-robust.

Next we prove that the language of subst-robust primitive words with one hole is closed under cyclic permutation. We know that two partial words x and y are conjugate if there exist partial words u and v such that $x \sqsubset uv$ and $y \sqsubset vu$. A language L is closed under conjugacy relation if the cyclic permutations of all the words are in L.

Lemma 21. *The language Q_p^{1S} is closed under conjugacy relation.*

Proof. We prove it by contradiction. Let $w = v_1v_2$ be a primitive partial word with one hole such that $w \in Q_p^{1S}$. Suppose $w' = v_2v_1 \notin Q_p^{1S}$. $w = v_1v_2 \in Q_p^{1S}$ implies $w = v_1v_2 \in Q_p$. Since $w' = v_2v_1 \notin Q_p^{1S}$ then we can write $w' = v_2v_1 \sqsubset u^{k_1}u_1au_2u^{k_2}$ such that $u_1bu_2 = u$. We consider two cases depending on whether a is in v_1 or in v_2.

Case A. If the symbol a in contained in v_2 then we consider the following possibilities.

Case A.1 If entire u_1au_2 is from v_2 then $v_2 \sqsubset u^{k_1}u_1au_2u^ru_1'$ and $v_1 \sqsubset u_2'u^s$ where $u = u_1'u_2'$ and $r + s + 1 = k_2$. Now $v_1v_2 \sqsubset u_2'u^su^{k_1}u_1au_2u^ru_1'$. Substituting a by b we obtain a nonprimitive partial word which is a contradiction that $v_1v_2 \in Q_p^{1S}$.

Case A.2 If a portion of u_2 is from v_2 then $v_2 \sqsubset u^{k_1}u_1au_2'$ and $v_1 \sqsubset u_2''u^{k_2}$ where $u_2 = u_2'u_2''$. Now $v_1v_2 \sqsubset u_2''u^{k_2}u^{k_1}u_1au_2'$ which will result a nonprimitive partial word after a is substituted by the letter b. Moreover $v_1v_2 \sqsubset (u_2''u_1bu_2')^{k_1+k_2+1}$ and v_1v_2 is not subst-robust primitive partial word.

Case B. If the symbol a is not contained in v_2 then we can handle two different cases as the previous one.

□

The following corollary is a consequence of Theorem 20.

Corollary 22. *A primitive partial word with one hole $w \in Q_p^{1\overline{S}}$ if and only if it is either contained in $u^nu'a$ or it's cyclic permutation for some $u \in Q_p$ and $u'b = u$ for $b \neq a$.*

Proof. The proof of necessary and sufficient conditions are as follow:
(Necessary Part:) Let $w = xay$ be a primitive partial word with one hole which is not subst-robust.Then $w = xay \sqsubset u^{k_1}u_1au_2u^{k_2}$ for some $u \in Q$ such that $u_1bu_2 = u$ and $k_1 + k_2 + 1 \geq 2$. As the language $Q_p^{1\overline{S}}$ is reflective, then $yxa \sqsubset u_2u^{k_2}u^{k_1}u_1a = (u_2u_1b)^{k_1+k_2}u_2u_1a = x^{k_1+k_2}x'a$ where $x = u_2u_2$ and $x' = u_2u_1$.

(Sufficient Part:) Let w be a partial word. If w is contained in $u^nu'a$ where $u'b = u$ or it's cyclic permutation then by substituting a by b where $b \neq a$, we obtain $w' \in u^{n+1}$ which is a nonprimitive partial word. Hence w is not a subst-robust primitive partial word.

6 Conclusion

We investigated exchange-robust and substitute robust primitive partial words with one hole. The structural characterization of each of the class of primitive partial words with one hole have been discussed and also some important combinatorial properties related to each of the class have been identified. We have shown that the language of non-exchange-robust primitive partial words is not

a context-free language. We mention some of the interesting questions that are still unanswered. (1) The notion of robustness can be studied further for partial words with at least two holes. (2) Is the language of primitive partial words Q_p^X context-free? (3) One can consider to exchange two symbols at any positions and preserve primitivity.

References

1. Berstel, J., Boasson, L.: Partial words and a theorem of Fine and Wilf. Theoretical Computer Science 218(1), 135–141 (1999)
2. Berstel, J., Perrin, D., et al.: Theory of codes, vol. 22. Citeseer (1985)
3. Blanchet-Sadri, F.: Primitive partial words. Discrete Applied Mathematics 148(3), 195–213 (2005)
4. Blanchet-Sadri, F.: Algorithmic combinatorics on partial words. CRC Press (2007)
5. Blanchet-Sadri, F., Nelson, S., Tebbe, A.: On operations preserving primitivity of partial words with one hole. In: AFL. pp. 93–107 (2011)
6. Crochemore, M., Rytter, W.: Jewels of stringology: text algorithms. World Scientific (2002)
7. Dassow, J., Martin, G.M., Vico, F.J.: Some operations preserving primitivity of words. Theoretical Computer Science 410(30), 2910–2919 (2009)
8. Dömösi, P., Ito, M.: Context-free languages and primitive words. World Scientific (2014)
9. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society 16(1), 109–114 (1965)
10. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
11. Lothaire, M.: Combinatorics on Words. Cambridge University Press (1997)
12. Lothaire, M.: Applied Combinatorics on Words. Cambridge University Press (2005)
13. Mitrana, V.: Primitive morphisms. Information processing letters 64(6), 277–281 (1997)
14. Nayak, A.C., Srivastava, A.K.: On del-robust primitive partial words with one hole. In: Language and Automata Theory and Applications, pp. 233–244. Springer (2016)
15. Ogden, W.: A helpful result for proving inherent ambiguity. Theory of Computing Systems 2(3), 191–194 (1968)
16. Păun, G., Santean, N., Thierrin, G., Yu, S.: On the robustness of primitive words. Discrete applied mathematics 117(1), 239–252 (2002)
17. Paun, G., Thierrin, G.: Morphisms and primitivity. Bulletin-European Association For Theoretical Computer Science 61, 85–88 (1997)
18. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages: Volume 1. Word, Language, Grammar, vol. 1. Springer (1997)
19. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008)

A Ogden's Lemma for CFL

For completeness we recall the Ogden's lemma to prove that the language of non-exchange-robust partial words is not context-free.

Lemma 23 (Ogden's Lemma [15]). *Let L be a context-free language. The there exists a constant $N > 0$ such that every string $w \in L$ that has at least N marked symbols can be decomposed in the form $w = uvxyz$ such that the following conditions hold:*

(i) together v and x have at least one marked symbol.
(ii) vxy contains at most N marked symbols.
(iii) $uv^ixy^iz \in L$ for all $i \geq 0$.

Disjunctive Probabilistic Modal Logic is Enough for Bisimilarity on Reactive Probabilistic Systems

Marco Bernardo[1] and Marino Miculan[2]

[1] Dip. di Scienze Pure e Applicate, Università di Urbino, Italy
[2] Dip. di Scienze Matematiche, Informatiche e Fisiche, Università di Udine, Italy

Abstract. Larsen and Skou characterized probabilistic bisimilarity over reactive probabilistic systems with a logic including true, negation, conjunction, and a diamond modality decorated with a probabilistic lower bound. Later on, Desharnais, Edalat, and Panangaden showed that negation is not necessary to characterize the same equivalence. In this paper, we prove that the logical characterization holds also when conjunction is replaced by disjunction, with negation still being not necessary. To this end, we introduce *reactive probabilistic trees*, a fully abstract model for reactive probabilistic systems that allows us to demonstrate expressiveness of the disjunctive probabilistic modal logic, as well as of the previously mentioned logics, by means of a compactness argument.

1 Introduction

Since its introduction [12], *probabilistic bisimilarity* has been used to compare probabilistic systems. It corresponds to Milner's *strong bisimilarity* for nondeterministic systems, and coincides with *lumpability* for Markov chains. Larsen and Skou [12] first proved that bisimilarity for *reactive probabilistic systems* can be given a *logical characterization*: two processes are bisimilar if and only if they satisfy the same set of formulas of a propositional modal logic similar to Hennessy-Milner logic [10]. In addition to the usual constructs $\top$, $\neg$, and $\wedge$, this logic features a diamond modality $\langle a \rangle_p \phi$, which is satisfied by a state if, after performing action a, the probability of being in a state satisfying ϕ is at least p.

Later on, Desharnais, Edalat, and Panangaden [6] showed that negation is *not* necessary for discrimination purposes, by working in a *continuous*-state setting. This result has no counterpart in the nonprobabilistic setting, where Hennessy-Milner logic without negation characterizes *simulation* equivalence, which is strictly coarser than bisimilarity [8] (while the two equivalences are known to coincide on reactive probabilistic systems [2]).

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 203–217 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

In this paper, we show that $\vee$ can be used in place of $\wedge$ without having to reintroduce negation: the constructs $\top$, $\vee$, and $\langle a \rangle_p$ suffice to characterize bisimilarity on reactive probabilistic systems. The intuition is that from a conjunctive distinguishing formula we can often derive a disjunctive one by suitably increasing some probabilistic lower bounds. Not even this result has a counterpart in the nonprobabilistic setting, where replacing conjunction with disjunction in the absence of negation yields trace equivalence (this equivalence does *not* coincide with bisimilarity on reactive probabilistic processes).

The proof of our result relies on a simple categorical construction of a semantics for reactive probabilistic systems, which we call *reactive probabilistic trees* (Sect. 3). This semantics is *fully abstract*, i.e., two states are probabilistically bisimilar if and only if they are mapped to the same reactive probabilistic tree. Moreover, the semantics is *compact*, in the sense that two (possibly infinite) trees are equal if and only if all of their *finite* approximations are equal. Hence, in order to prove that a logic characterizes probabilistic bisimilarity, it suffices to prove that it allows to discriminate finite reactive probabilistic trees. Indeed, given two different finite trees, we can construct a formula of the considered logic (by induction on the height of one of the trees) that tells the two trees apart and has a depth not exceeding the height of the two trees (Sect. 4). Our technique applies also to the logics in [12,6], for which it allows us to provide simpler proofs of adequacy, directly in a *discrete* setting. More generally, this technique can be used in any computational model that has a compact, fully abstract semantics.

2 Processes, Bisimilarity, and Logics

2.1 Reactive Probabilistic Processes and Strong Bisimilarity

Probabilistic processes can be represented as labeled transitions systems with probabilistic information used to determine which action is executed or which state is reached. Following the terminology of [9], we focus on *reactive* probabilistic processes, where every state has for each action at most one outgoing distribution over states; the choice among these arbitrarily many, differently labeled distributions is nondeterministic. For a countable (i.e., finite or countably infinite) set X, the set of *finitely supported* (a.k.a. simple) probability distributions over X is given by $D(X) = \{\Delta : X \to \mathbb{R}_{[0,1]} \mid |\mathit{supp}(\Delta)| < \omega, \sum_{x \in X} \Delta(x) = 1\}$, where the *support* of distribution Δ is defined as $\mathit{supp}(\Delta) \triangleq \{x \in X \mid \Delta(x) > 0\}$.

A *reactive probabilistic labeled transition system*, RPLTS for short, is a triple $(S, A, \longrightarrow)$ where S is a countable set of *states*, A is a countable set of *actions*, and $\longrightarrow \subseteq S \times A \times D(S)$ is a *transition relation* such that, whenever $(s, a, \Delta_1), (s, a, \Delta_2) \in \longrightarrow$, then $\Delta_1 = \Delta_2$.

An RPLTS can be seen as a directed graph whose edges are labeled by pairs $(a, p) \in A \times \mathbb{R}_{(0,1]}$. For every $s \in S$ and $a \in A$, if there are a-labeled edges outgoing from s, then these are finitely many (*image finiteness*), because the considered distributions are finitely supported, and the numbers on them add up to 1. As usual, we denote $(s, a, \Delta) \in \longrightarrow$ as $s \xrightarrow{a} \Delta$, where the set of

reachable states coincides with $supp(\Delta)$. We also define cumulative reachability as $\Delta(S') = \sum_{s' \in S'} \Delta(s')$ for all $S' \subseteq S$.

Probabilistic bisimilarity for the class of reactive probabilistic processes was introduced by Larsen and Skou [12]. Let $(S, A, \longrightarrow)$ be an RPLTS. An equivalence relation $\mathcal{B}$ over S is a *probabilistic bisimulation* iff, whenever $(s_1, s_2) \in \mathcal{B}$, then for all actions $a \in A$ it holds that, if $s_1 \xrightarrow{a} \Delta_1$, then $s_2 \xrightarrow{a} \Delta_2$ and $\Delta_1(C) = \Delta_2(C)$ for all equivalence classes $C \in S/\mathcal{B}$. We say that $s_1, s_2 \in S$ are *probabilistically bisimilar*, written $s_1 \sim_{\mathrm{PB}} s_2$, iff there exists a probabilistic bisimulation including the pair (s_1, s_2).

2.2 Probabilistic Modal Logics

In our setting, a probabilistic modal logic is a pair formed by a set $\mathcal{L}$ of *formulas* and an RPLTS-indexed family of *satisfaction relations* $\models \subseteq S \times \mathcal{L}$. The *logical equivalence* induced by $\mathcal{L}$ over S is defined by letting $s_1 \cong_{\mathcal{L}} s_2$, where $s_1, s_2 \in S$, iff $s_1 \models \phi \iff s_2 \models \phi$ for all $\phi \in \mathcal{L}$. We say that $\mathcal{L}$ *characterizes* a binary relation $\mathcal{R}$ over S when $\mathcal{R} = \cong_{\mathcal{L}}$.

We are especially interested in probabilistic modal logics characterizing $\sim_{\mathrm{PB}}$. The logics considered in this paper are similar to Hennessy-Milner logic [10], but the diamond modality is decorated with a probabilistic lower bound as follows:

$$\mathrm{PML}_{\neg\wedge} : \phi ::= \top \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle_p \phi \qquad \mathrm{PML}_{\wedge} : \phi ::= \top \mid \phi \wedge \phi \mid \langle a \rangle_p \phi$$
$$\mathrm{PML}_{\neg\vee} : \phi ::= \top \mid \neg\phi \mid \phi \vee \phi \mid \langle a \rangle_p \phi \qquad \mathrm{PML}_{\vee} : \phi ::= \top \mid \phi \vee \phi \mid \langle a \rangle_p \phi$$

where $p \in \mathbb{R}_{[0,1]}$; trailing $\top$'s will be omitted for sake of readability. Their semantics with respect to an RPLTS state s is defined as usual, in particular:

$$s \models \langle a \rangle_p \phi \iff s \xrightarrow{a} \Delta \text{ and } \Delta(\{s' \in S \mid s' \models \phi\}) \geq p$$

Larsen and Skou [12] proved that $\mathrm{PML}_{\neg\wedge}$ (and hence $\mathrm{PML}_{\neg\vee}$) characterizes $\sim_{\mathrm{PB}}$. Desharnais, Edalat, and Panangaden [6] then proved in a *measure-theoretic* setting that $\mathrm{PML}_{\wedge}$ characterizes $\sim_{\mathrm{PB}}$ too, and hence negation is not necessary. This was subsequently redemonstrated by Jacobs and Sokolova [11] in the *dual adjunction* framework and by Deng and Wu [5] for a *fuzzy* extension of RPLTS. The main aim of this paper is to show that $\mathrm{PML}_{\vee}$ suffices as well.

3 Compact Characterization of Probabilistic Bisimilarity

3.1 Coalgebras for Probabilistic Systems

It is well known that the function D defined in Sect. 2.1 extends to a functor $D : \mathsf{Set} \to \mathsf{Set}$ whose action on morphisms is, for $f : X \to Y$, $D(f)(\Delta) = \lambda y.\Delta(f^{-1}(y))$. Then, every RPLTS corresponds to a coalgebra of the functor $B_{RP} : \mathsf{Set} \to \mathsf{Set}, B_{RP}(X) \triangleq (D(X)+1)^A$. Indeed, for $S = (S, A, \longrightarrow)$, the corresponding coalgebra $(S, \sigma : S \to B_{RP}(S))$ is $\sigma(s) \triangleq \lambda a.(\text{if } s \xrightarrow{a} \Delta \text{ then } \Delta \text{ else } *)$. A *homomorphism* $h : (S, \sigma) \to (T, \tau)$ is a function $h : S \to T$ that respects the coalgebraic structures, i.e., $\tau \circ h = (B_{RP}h) \circ \sigma$. We denote by $Coalg(B_{RP})$ the category of B_{RP}-coalgebras and their homomorphisms.

Aczel and Mendler [1] introduced a general notion of bisimulation for coalgebras, which in our setting instantiates as follows:

Definition 1. *Let (S_1, σ_1), (S_2, σ_2) be B_{RP}-coalgebras. A relation $\mathcal{R} \subseteq S_1 \times S_2$ is a B_{RP}-*bisimulation *iff there exists a coalgebra structure $\rho : \mathcal{R} \to B_{RP}\mathcal{R}$ such that the projections $\pi_1 : \mathcal{R} \to S_1$, $\pi_2 : \mathcal{R} \to S_2$ are homomorphisms (i.e., $\sigma_i \circ \pi_i = B_{RP}\pi_i \circ \rho$ for $i = 1, 2$). We say that $s_1 \in S_1$, $s_2 \in S_2$ are B_{RP}-*bisimilar*, written $s_1 \sim s_2$, iff there exists a B_{RP}-bisimulation including (s_1, s_2).*

Proposition 1. *The probabilistic bisimilarity over an RPLTS $(S, A, \longrightarrow)$ coincides with the B_{RP}-bisimilarity over the corresponding coalgebra (S, σ).*

B_{RP} is finitary (because we restrict to finitely supported distributions) and hence admits final coalgebra (cf. [3,15] and specifically [14, Thm. 4.6]). The final coalgebra is unique up-to isomorphism, and can be seen as the RPLTS whose elements are canonical representatives of all possible behaviors of any RPLTS:

Proposition 2. *Let (Z, ζ) be a final B_{RP}-coalgebra. For all $z_1, z_2 \in Z$: $z_1 \sim z_2$ iff $z_1 = z_2$.*

3.2 Reactive Probabilistic Trees

We now introduce *reactive probabilistic trees*, a representation of the final B_{RP}-coalgebra that can be seen as the natural extension to the probabilistic setting of *strongly extensional trees* used to represent the final $\mathcal{P}_f$-coalgebra [15].

Definition 2 (*RPT*). *An* (A-labeled) reactive probabilistic tree *is a pair* $(X, succ)$ *where $X \in \mathsf{Set}$ and $succ : X \times A \to \mathcal{P}_f(X \times \mathbb{R}_{(0,1]})$ are such that the relation $\leq$ over X, defined by the rules $\frac{}{x \leq x}$ and $\frac{x \leq y \quad z \in succ(y,a)}{x \leq z}$, is a partial order with a least element, called* root*, and for all $x \in X$ and $a \in A$:*

1. *the set $\{y \in X \mid y \leq x\}$ is finite and well-ordered;*
2. *for all $(x_1, p_1), (x_2, p_2) \in succ(x, a)$: if $x_1 = x_2$ then $p_1 = p_2$; if the subtrees rooted at x_1 and x_2 are isomorphic then $x_1 = x_2$;*
3. *if $succ(x, a) \neq \emptyset$ then $\sum_{(y,p) \in succ(x,a)} p = 1$.*

Reactive probabilistic trees are unordered trees where each node for each action has either no successors or a finite set of successors, which are labeled with positive real numbers that add up to 1; moreover, subtrees rooted at these successors are all *different*. See the forthcoming Fig. 1 for some examples. In particular, the trivial tree is $nil \triangleq (\{\bot\}, \lambda x, a.\emptyset)$.

We denote by RPT, ranged over by t, t_1, t_2, the set of reactive probabilistic trees (possibly of infinite height), up-to isomorphism. For $t = (X, succ)$, we denote its root by $\bot_t$, its a-successors by $t(a) \triangleq succ(\bot_t, a)$, and the subtree rooted at $x \in X$ by $t[x] \triangleq (\{y \in X \mid x \leq y\}, \lambda y, a.succ(y, a))$; thus, $\bot_{t[x]} = x$. We define $height : RPT \to \mathbb{N} \cup \{\omega\}$ as $height(t) \triangleq \sup\{1 + height(t') \mid (t', p) \in t(a), a \in A\}$ with $\sup \emptyset = 0$; hence, $height(nil) = 0$. We denote by $RPT_f \triangleq \{t \in RPT \mid height(t) < \omega\}$ the set of reactive probabilistic trees of finite height.

A (possibly infinite) tree can be *pruned* at any height n, yielding a finite tree where the removed subtrees are replaced by *nil*. The "pruning" function

$(\cdot)|_n : RPT \to RPT_f$, parametric in n, can be defined by first truncating the tree t at height n, and then collapsing isomorphic subtrees adding their weights.

We have now to show that RPT is (the carrier of) the final B_{RP}-coalgebra (up-to isomorphism). To this end, we reformulate B_{RP} in a slightly more "relational" format. We define a functor $D' : \mathsf{Set} \to \mathsf{Set}$ as follows:

$$D'X \triangleq \{\emptyset\} \cup \{U \in \mathcal{P}_f(X \times \mathbb{R}_{(0,1]}) \mid \textstyle\sum_{(x,p)\in U} p = 1 \text{ and } (x,p),(x,q) \in U \Rightarrow p = q\}$$
$$D'f \triangleq \lambda U \in D'X.\{(f(x), \textstyle\sum_{(x,p)\in U} p) \mid x \in \pi_1(U)\} \qquad \text{for any } f : X \to Y.$$

Proposition 3. *$D'^A \cong B_{RP}$, and $Coalg(D'^A) \cong Coalg(B_{RP})$; hence the (supports of the) final D'^A-coalgebra and the final B_{RP}-coalgebra are isomorphic.*

RPT is the carrier of the final B_{RP}-coalgebra (up-to isomorphism). In fact, RPT can be endowed with a D'^A-coalgebra structure $\rho : RPT \to (D'(RPT))^A$ defined, for $t = (X, succ)$, as $\rho(t)(a) \triangleq \{(t[x], p) \mid (x,p) \in succ(\bot_t, a)\}$.

Theorem 1. *(RPT, ρ) is a final B_{RP}-coalgebra.*

By virtue of Thm. 1, given an RPLTS $S = (S, A, \longrightarrow)$ there exists a unique coalgebra homomorphism $[\![\cdot]\!] : S \to RPT$, called the *(final) semantics* of S, which associates each state in S with its behavior. This semantics is *fully abstract*. Another key property of reactive probabilistic trees is that they are *compact*: two different trees can be distinguished by looking at their finite subtrees only.

Theorem 2 (Full abstraction). *Let $(S, A, \longrightarrow)$ be an RPLTS. For all $s_1, s_2 \in S$: $s_1 \sim_{\mathrm{PB}} s_2$ iff $[\![s_1]\!] = [\![s_2]\!]$.*

Theorem 3 (Compactness). *For all $t_1, t_2 \in RPT$: $t_1 = t_2$ iff for all $n \in \mathbb{N}$: $t_1|_n = t_2|_n$.*

Corollary 1. *Let $(S, A, \longrightarrow)$ be an RPLTS. For all $s_1, s_2 \in S$: $s_1 \sim_{\mathrm{PB}} s_2$ iff for all $n \in \mathbb{N} : [\![s_1]\!]|_n = [\![s_2]\!]|_n$.*

4 The Discriminating Power of $\mathrm{PML}_\vee$

By virtue of the categorical construction leading to Cor. 1, in order to prove that a modal logic characterizes $\sim_{\mathrm{PB}}$ over reactive probabilistic processes, it is enough to show that it can discriminate all reactive probabilistic trees of *finite* height. A specific condition on the depth of distinguishing formulas has also to be satisfied, where $depth(\phi)$ is defined as usual:

$$depth(\top) = 0 \qquad depth(\neg\phi') = depth(\phi') \qquad depth(\langle a\rangle_p\phi') = 1 + depth(\phi')$$
$$depth(\phi_1 \wedge \phi_2) = depth(\phi_1 \vee \phi_2) = \max(depth(\phi_1), depth(\phi_2))$$

Proposition 4. *Let $\mathcal{L}$ be one of the probabilistic modal logics in Sect. 2.2. If $\mathcal{L}$ characterizes $=$ over RPT_f and for any two nodes t_1 and t_2 of an arbitrary RPT_f model such that $t_1 \neq t_2$ there exists $\phi \in \mathcal{L}$ distinguishing t_1 from t_2 such that $depth(\phi) \leq \max(height(t_1), height(t_2))$, then $\mathcal{L}$ characterizes $\sim_{\mathrm{PB}}$ over the set of RPLTS models.*

In this section, we show the main result of the paper: the logical equivalence induced by $\mathrm{PML}_{\vee}$ has the same discriminating power as $\sim_{\mathrm{PB}}$. This result is accomplished in three steps. Firstly, we redemonstrate Larsen and Skou's result for $\mathrm{PML}_{\neg\wedge}$ in the RPT_f setting, which yields a proof that, with respect to the one in [12], is simpler and does not require the minimal deviation assumption (i.e., that the probability associated with any state in the support of the target distribution of a transition be a multiple of some value). This provides a proof scheme for the subsequent steps. Secondly, we demonstrate that $\mathrm{PML}_{\neg\vee}$ characterizes $\sim_{\mathrm{PB}}$ by adapting the proof scheme to cope with the replacement of $\wedge$ with $\vee$. Thirdly, we demonstrate that $\mathrm{PML}_{\vee}$ characterizes $\sim_{\mathrm{PB}}$ by further adapting the proof scheme to cope with the absence of $\neg$.

Moreover, we redemonstrate Desharnais, Edalat, and Panangaden's result for $\mathrm{PML}_{\wedge}$ through yet another adaptation of the proof scheme that, unlike the proof in [6], works directly on *discrete* state spaces without making use of measure-theoretic arguments. Avoiding the resort to measure theory was shown to be possible for the first time by Worrell in an unpublished note cited in [13].

4.1 $\mathrm{PML}_{\neg\wedge}$ Characterizes $\sim_{\mathrm{PB}}$: A New Proof

To show that the logical equivalence induced by $\mathrm{PML}_{\neg\wedge}$ implies node equality $=$, we reason on the contrapositive. Given two nodes t_1 and t_2 such that $t_1 \neq t_2$, we proceed by induction on the height of t_1 to find a distinguishing $\mathrm{PML}_{\neg\wedge}$ formula whose depth is not greater than the heights of t_1 and t_2. The idea is to exploit negation, so to ensure that certain distinguishing formulas are *satisfied* by a certain derivative t' of t_1 (rather than the derivatives of t_2 different from t'), then take the *conjunction* of those formulas preceded by a diamond decorated with the probability for t_1 of *reaching* t'.

The only non-trivial case is the one in which t_1 and t_2 enable the same actions. At least one of those actions, say a, is such that, after performing it, the two nodes reach two distributions $\Delta_{1,a}$ and $\Delta_{2,a}$ such that $\Delta_{1,a} \neq \Delta_{2,a}$. Given a node $t' \in \mathit{supp}(\Delta_{1,a})$ such that $\Delta_{1,a}(t') > \Delta_{2,a}(t')$, by the induction hypothesis there exists a $\mathrm{PML}_{\neg\wedge}$ formula $\phi'_{2,j}$ that distinguishes t' from a specific $t'_{2,j} \in \mathit{supp}(\Delta_{2,a}) \setminus \{t'\}$. We can assume that $t' \models \phi'_{2,j} \not\models t'_{2,j}$ otherwise, thanks to the presence of negation in $\mathrm{PML}_{\neg\wedge}$, it would suffice to consider $\neg\phi'_{2,j}$.

As a consequence, $t_1 \models \langle a \rangle_{\Delta_{1,a}(t')} \bigwedge_j \phi'_{2,j} \not\models t_2$ because $\Delta_{1,a}(t') > \Delta_{2,a}(t')$ and $\Delta_{2,a}(t')$ is the maximum probabilistic lower bound for which t_2 satisfies a formula of that form. Notice that $\Delta_{1,a}(t')$ may not be the maximum probabilistic lower bound for which t_1 satisfies such a formula, because $\bigwedge_j \phi'_{2,j}$ might be satisfied by other a-derivatives of t_1 in $\mathit{supp}(\Delta_{1,a}) \setminus \{t'\}$.

Theorem 4. *Let $(T, A, \longrightarrow)$ be in RPT_f and $t_1, t_2 \in T$. Then $t_1 = t_2$ iff $t_1 \models \phi \iff t_2 \models \phi$ for all $\phi \in \mathrm{PML}_{\neg\wedge}$. Moreover, if $t_1 \neq t_2$, then there exists $\phi \in \mathrm{PML}_{\neg\wedge}$ distinguishing t_1 from t_2 such that $\mathit{depth}(\phi) \leq \max(\mathit{height}(t_1), \mathit{height}(t_2))$.*

4.2 $\mathrm{PML}_{\neg\vee}$ Characterizes $\sim_{\mathrm{PB}}$: Adapting the Proof

Since $\phi_1 \wedge \phi_2$ is logically equivalent to $\neg(\neg\phi_1 \vee \neg\phi_2)$, it is not surprising that $\mathrm{PML}_{\neg\vee}$ characterizes $\sim_{\mathrm{PB}}$ too. However, the proof of this result will be useful to

set up an outline of the proof of the main result of this paper, i.e., that $\text{PML}_{\vee}$ characterizes $\sim_{\text{PB}}$ as well.

Similar to the proof of Thm. 4, also for $\text{PML}_{\neg\vee}$ we reason on the contrapositive and proceed by induction. Given t_1 and t_2 such that $t_1 \not\sim t_2$, we intend to exploit negation, so to ensure that certain distinguishing formulas are *not satisfied* by a certain derivative t' of t_1 (rather than the derivatives of t_2 different from t'), then take the *disjunction* of those formulas preceded by a diamond decorated with the probability for t_2 of *not reaching* t'.

In the only non-trivial case, for $t' \in \mathit{supp}(\Delta_{1,a})$ such that $\Delta_{1,a}(t') > \Delta_{2,a}(t')$, by the induction hypothesis there exists a $\text{PML}_{\neg\vee}$ formula $\phi'_{2,j}$ that distinguishes t' from a specific $t'_{2,j} \in \mathit{supp}(\Delta_{2,a}) \setminus \{t'\}$. We can assume that $t' \not\models \phi'_{2,j} \not\ni t'_{2,j}$ otherwise, thanks to the presence of negation in $\text{PML}_{\neg\vee}$, it would suffice to consider $\neg\phi'_{2,j}$. Therefore, $t_1 \not\models \langle a \rangle_{1-\Delta_{2,a}(t')} \bigvee_j \phi'_{2,j} \not\ni t_2$ because $1-\Delta_{2,a}(t') > 1-\Delta_{1,a}(t')$ and the maximum probabilistic lower bound for which t_1 satisfies a formula of that form cannot exceed $1 - \Delta_{1,a}(t')$. Notice that $1 - \Delta_{2,a}(t')$ is the *maximum* probabilistic lower bound for which t_2 satisfies such a formula, because that value is the probability with which t_2 does not reach t' after performing a.

Theorem 5. *Let $(T, A, \longrightarrow)$ be in RPT_f and $t_1, t_2 \in T$. Then $t_1 = t_2$ iff $t_1 \models \phi \iff t_2 \models \phi$ for all $\phi \in \mathit{PML}_{\neg\vee}$. Moreover, if $t_1 \not\sim t_2$, then there exists $\phi \in \mathit{PML}_{\neg\vee}$ distinguishing t_1 from t_2 such that $\mathit{depth}(\phi) \leq \max(\mathit{height}(t_1), \mathit{height}(t_2))$.*

4.3 Also $\text{PML}_{\vee}$ Characterizes $\sim_{\text{PB}}$

The proof that $\text{PML}_{\vee}$ characterizes $\sim_{\text{PB}}$ is inspired by the one for $\text{PML}_{\neg\vee}$, thus considers the contrapositive and proceeds by induction. In the only non-trivial case, we will arrive at a point in which $t_1 \not\models \langle a \rangle_{1-(\Delta_{2,a}(t')+p)} \bigvee_{j \in J} \phi'_{2,j} \not\ni t_2$ for:

- a derivative t' of t_1, such that $\Delta_{1,a}(t') > \Delta_{2,a}(t')$, not satisfying any subformula $\phi'_{2,j}$;
- a suitable probabilistic value p such that $\Delta_{2,a}(t') + p < 1$;
- an index set J identifying certain derivatives of t_2 other than t'.

The choice of t' is crucial, because negation is no longer available in $\text{PML}_{\vee}$. Different from the case of $\text{PML}_{\neg\vee}$, this induces the introduction of p and the limitation to J in the format of the distinguishing formula. An important observation is that, in many cases, a disjunctive distinguishing formula can be obtained from a conjunctive one by suitably *increasing* some probabilistic lower bounds. An obvious exception is when the use of conjunction/disjunction is not necessary for telling two different nodes apart.

Example 1. The nodes t_1 and t_2 in Fig. 1(a) cannot be distinguished by any formula in which neither conjunction nor disjunction occurs. It holds that:

$$t_1 \models \langle a \rangle_{0.5} (\langle b \rangle_1 \wedge \langle c \rangle_1) \not\models t_2 \qquad t_1 \not\models \langle a \rangle_{1.0} (\langle b \rangle_1 \vee \langle c \rangle_1) \not\ni t_2$$

Notice that, when moving from the conjunctive formula to the disjunctive one, the probabilistic lower bound decorating the a-diamond increases from 0.5 to 1 and the roles of t_1 and t_2 with respect to $\models$ are inverted. The situation is similar for

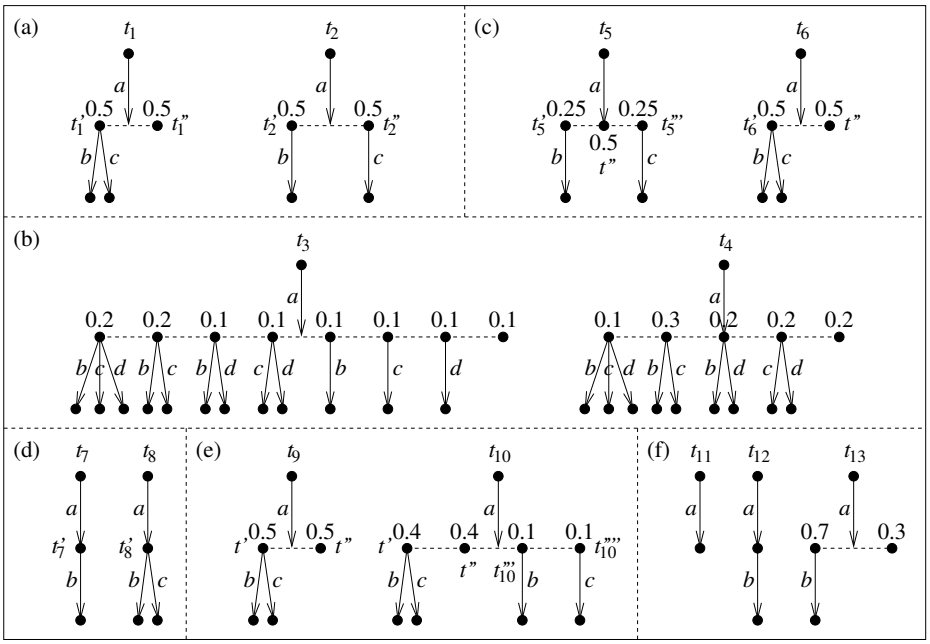


Fig. 1. RPT_f models used in the examples of Sects. 4.3 and 4.4.

the nodes t_3 and t_4 in Fig. 1(b), where two occurrences of conjunction/disjunction are necessary:
$t_3 \models \langle a \rangle_{0.2} (\langle b \rangle_1 \wedge \langle c \rangle_1 \wedge \langle d \rangle_1) \not\models t_4 \qquad t_3 \models \langle a \rangle_{0.9} (\langle b \rangle_1 \vee \langle c \rangle_1 \vee \langle d \rangle_1) \not\models t_4$
but the roles of t_3 and t_4 with respect to $\models$ cannot be inverted. ■

Example 2. For the nodes t_5 and t_6 in Fig. 1(c), it holds that:

$$t_5 \not\models \langle a \rangle_{0.5} (\langle b \rangle_1 \wedge \langle c \rangle_1) \not\dashv t_6$$

If we replace conjunction with disjunction and we vary the probabilistic lower bound between 0.5 and 1, we produce no disjunctive formula capable of discriminating between t_5 and t_6. Nevertheless, a distinguishing formula belonging to $\mathrm{PML}_\vee$ exists with no disjunctions at all:

$$t_5 \not\models \langle a \rangle_{0.5} \langle b \rangle_1 \not\dashv t_6$$ ■

The examples above show that the increase of some probabilistic lower bounds when moving from conjunctive distinguishing formulas to disjunctive ones takes place only in the case that the probabilities of reaching certain nodes have to be *summed up*. Additionally, we recall that, in order for two nodes to be related by $\sim_{\mathrm{PB}}$, they must enable the same actions, so focussing on a *single* action is enough for discriminating when only disjunction is available. Bearing this in mind, for any node t of finite height we define the set $\Phi_\vee(t)$ of $\mathrm{PML}_\vee$ formulas satisfied by t featuring:

- probabilistic lower bounds of diamonds that are *maximal* with respect to the satisfiability of a formula of that format by t (this is consistent with the observation in the last sentence before Thm. 5, and keeps the set $\Phi_\vee(t)$ finite);
- diamonds that arise only from *existing* transitions that depart from t (so to avoid useless diamonds in disjunctions and hence keep the set $\Phi_\vee(t)$ finite);
- disjunctions that stem only from *single* transitions of *different* nodes in the support of a distribution reached by t (transitions departing from the same node would result in formulas like $\bigvee_{h\in H}\langle a_h\rangle_{p_h}\phi_h$, with $a_{h_1}\neq a_{h_2}$ for $h_1\neq h_2$, which are useless for discriminating with respect to $\sim_{\mathrm{PB}}$) and are preceded by a diamond decorated with the *sum* of the probabilities assigned to those nodes by the distribution reached by t.

Definition 3. *The set $\Phi_\vee(t)$ for a node t of finite height is defined by induction on height(t) as follows:*

- *If height$(t)=0$, then $\Phi_\vee(t)=\emptyset$.*
- *If height$(t)\geq 1$ for t having transitions of the form $t\xrightarrow{a_i}\Delta_i$ with supp$(\Delta_i)=\{t'_{i,j}\mid j\in J_i\}$ and $i\in I\neq\emptyset$, then: $\Phi_\vee(t)=\{\langle a_i\rangle_1\mid i\in I\}\cup$*

$$\bigcup_{i\in I} hplb\Big(\bigcup_{\emptyset\neq J'\subseteq J_i}\Big\{\langle a_i\rangle \sum_{j\in J'}\Delta_i(t'_{i,j})\ \dot\bigvee_{j\in J'}\phi'_{i,j,k}\mid t'_{i,j}\in supp(\Delta_i),\phi'_{i,j,k}\in\Phi_\vee(t'_{i,j})\Big\}\Big)$$

where $\dot\vee$ is a variant of $\vee$ in which identical operands are not admitted (i.e., idempotence is forced) and hplb keeps only the formula with the highest probabilistic lower bound decorating the initial a_i-diamond among the formulas differring only for that bound.

To illustrate the definition given above, we exhibit some examples showing the usefulness of $\Phi_\vee$-sets for discrimination purposes. Given two different nodes that with the same action reach two different distributions, a good criterion for choosing t' (a derivative of the first node not satisfying certain formulas, to which the first distribution assigns a probability greater than the second one) seems to be the *minimality* of the $\Phi_\vee$-set.

Example 3. For the nodes t_7 and t_8 in Fig. 1(d), we have:

$$\Phi_\vee(t_7)=\{\langle a\rangle_1,\langle a\rangle_1\langle b\rangle_1\}\qquad \Phi_\vee(t_8)=\{\langle a\rangle_1,\langle a\rangle_1\langle b\rangle_1,\langle a\rangle_1\langle c\rangle_1\}$$

A formula like $\langle a\rangle_1(\langle b\rangle_1\vee\langle c\rangle_1)$ is useless for discriminating between t_7 and t_8, because disjunction is between two actions enabled by the same node and hence constituting a nondeterministic choice. Indeed, such a formula is not part of $\Phi_\vee(t_8)$. While in the case of conjunction it is often necessary to concentrate on several alternative actions, in the case of disjunction it is convenient to focus on a single action per node when aiming at producing a distinguishing formula.

The fact that $\langle a\rangle_1\langle c\rangle_1\in\Phi_\vee(t_8)$ is a distinguishing formula can be retrieved as follows. Starting from the two identically labeled transitions $t_7\xrightarrow{a}\Delta_{7,a}$ and $t_8\xrightarrow{a}\Delta_{8,a}$ where $\Delta_{7,a}(t'_7)=1=\Delta_{8,a}(t'_8)$ and $\Delta_{7,a}(t'_8)=0=\Delta_{8,a}(t'_7)$, we have:

$$\Phi_\vee(t'_7)=\{\langle b\rangle_1\}\qquad \Phi_\vee(t'_8)=\{\langle b\rangle_1,\langle c\rangle_1\}$$

If we focus on t'_7 because $\Delta_{7,a}(t'_7)>\Delta_{8,a}(t'_7)$ and its $\Phi_\vee$-set is minimal, then $t'_7\not\models\langle c\rangle_1\mathrel{=\!\!|}t'_8$ with $\langle c\rangle_1\in\Phi_\vee(t'_8)\setminus\Phi_\vee(t'_7)$. As a consequence, $t_7\not\models\langle a\rangle_1\langle c\rangle_1\mathrel{=\!\!|}t_8$ where the value 1 decorating the a-diamond stems from $1-\Delta_{8,a}(t'_7)$. ■

Example 4. For the nodes t_1 and t_2 in Fig. 1(a), we have:

$$\Phi_\vee(t_1) = \{\langle a\rangle_1,\ \langle a\rangle_{0.5}\langle b\rangle_1,\ \langle a\rangle_{0.5}\langle c\rangle_1\}$$
$$\Phi_\vee(t_2) = \{\langle a\rangle_1,\ \langle a\rangle_{0.5}\langle b\rangle_1,\ \langle a\rangle_{0.5}\langle c\rangle_1,\ \langle a\rangle_1(\langle b\rangle_1 \vee \langle c\rangle_1)\}$$

The formulas with two diamonds and no disjunction are identical in the two sets, so their disjunction $\langle a\rangle_{0.5}\langle b\rangle_1 \vee \langle a\rangle_{0.5}\langle c\rangle_1$ is useless for discriminating between t_1 and t_2. Indeed, such a formula is part of neither $\Phi_\vee(t_1)$ nor $\Phi_\vee(t_2)$. In contrast, their disjunction in which decorations of identical diamonds are summed up, i.e., $\langle a\rangle_1(\langle b\rangle_1 \vee \langle c\rangle_1)$, is fundamental. It belongs only to $\Phi_\vee(t_2)$ because in the case of t_1 the b-transition and the c-transition depart from the same node, hence no probabilities can be added.

The fact that $\langle a\rangle_1(\langle b\rangle_1 \vee \langle c\rangle_1) \in \Phi_\vee(t_2)$ is a distinguishing formula can be retrieved as follows. Starting from the two identically labeled transitions $t_1 \xrightarrow{a} \Delta_{1,a}$ and $t_2 \xrightarrow{a} \Delta_{2,a}$ where $\Delta_{1,a}(t_1') = \Delta_{1,a}(t_1'') = 0.5 = \Delta_{2,a}(t_2') = \Delta_{2,a}(t_2'')$ and $\Delta_{1,a}(t_2') = \Delta_{1,a}(t_2'') = 0 = \Delta_{2,a}(t_1') = \Delta_{2,a}(t_1'')$, we have:

$$\Phi_\vee(t_1') = \{\langle b\rangle_1, \langle c\rangle_1\} \quad \Phi_\vee(t_1'') = \emptyset \quad \Phi_\vee(t_2') = \{\langle b\rangle_1\} \quad \Phi_\vee(t_2'') = \{\langle c\rangle_1\}$$

If we focus on t_1'' because $\Delta_{1,a}(t_1'') > \Delta_{2,a}(t_1'')$ and its $\Phi_\vee$-set is minimal, then $t_1'' \not\models \langle b\rangle_1 \rightleftharpoons t_2'$ with $\langle b\rangle_1 \in \Phi_\vee(t_2') \setminus \Phi_\vee(t_1'')$ as well as $t_1'' \not\models \langle c\rangle_1 \rightleftharpoons t_2''$ with $\langle c\rangle_1 \in \Phi_\vee(t_2'') \setminus \Phi_\vee(t_1'')$. Thus, $t_1 \not\models \langle a\rangle_1(\langle b\rangle_1 \vee \langle c\rangle_1) \rightleftharpoons t_2$ where value 1 decorating the a-diamond stems from $1 - \Delta_{2,a}(t_1'')$. ■

Example 5. For the nodes t_5 and t_6 in Fig. 1(c), we have:

$$\Phi_\vee(t_5) = \{\langle a\rangle_1,\ \langle a\rangle_{0.25}\langle b\rangle_1,\ \langle a\rangle_{0.25}\langle c\rangle_1,\ \langle a\rangle_{0.5}(\langle b\rangle_1 \vee \langle c\rangle_1)\}$$
$$\Phi_\vee(t_6) = \{\langle a\rangle_1,\ \langle a\rangle_{0.5}\langle b\rangle_1,\ \langle a\rangle_{0.5}\langle c\rangle_1\}$$

The formulas with two diamonds and no disjunction are different in the two sets, so they are enough for discriminating between t_5 and t_6. In contrast, the only formula with disjunction, occurring in $\Phi_\vee(t_5)$, is useless because the probability decorating its a-diamond is equal to the one decorating the a-diamond of each of the two formulas with two diamonds in $\Phi_\vee(t_6)$.

The fact that $\langle a\rangle_{0.5}\langle b\rangle_1 \in \Phi_\vee(t_6)$ is a distinguishing formula can be retrieved as follows. Starting from the two identically labeled transitions $t_5 \xrightarrow{a} \Delta_{5,a}$ and $t_6 \xrightarrow{a} \Delta_{6,a}$ where $\Delta_{5,a}(t_5') = \Delta_{5,a}(t_5''') = 0.25$, $\Delta_{5,a}(t'') = 0.5 = \Delta_{6,a}(t_6') = \Delta_{6,a}(t'')$, and $\Delta_{5,a}(t_6') = 0 = \Delta_{6,a}(t_5') = \Delta_{6,a}(t_5''')$, we have:

$$\Phi_\vee(t_5') = \{\langle b\rangle_1\} \quad \Phi_\vee(t_5''') = \{\langle c\rangle_1\} \quad \Phi_\vee(t_6') = \{\langle b\rangle_1, \langle c\rangle_1\} \quad \Phi_\vee(t'') = \emptyset$$

Notice that t'' might be useless for discriminating purposes because it has the same probability in both distributions, so we exclude it. If we focus on t_5''' because $\Delta_{5,a}(t_5''') > \Delta_{6,a}(t_5''')$ and its $\Phi_\vee$-set is minimal after the exclusion of t'', then $t_5''' \not\models \langle b\rangle_1 \rightleftharpoons t_6'$ with $\langle b\rangle_1 \in \Phi_\vee(t_6') \setminus \Phi_\vee(t_5''')$, while no distinguishing formula is considered with respect to t'' as element of $supp(\Delta_{6,a})$ due to the exclusion of t'' itself. As a consequence, $t_5 \not\models \langle a\rangle_{0.5}\langle b\rangle_1 \rightleftharpoons t_6$ where the value 0.5 decorating the a-diamond stems from $1 - (\Delta_{6,a}(t_5''') + p)$ with $p = \Delta_{6,a}(t'')$. The reason for subtracting the probability that t_6 reaches t'' after performing a is that $t'' \not\models \langle b\rangle_1$.

We conclude by observing that focussing on t'' as derivative with the minimum $\Phi_\vee$-set is indeed problematic, because it would result in $\langle a\rangle_{0.5}\langle b\rangle_1$ when considering t'' as derivative of t_5, but it would result in $\langle a\rangle_{0.5}(\langle b\rangle_1 \vee \langle c\rangle_1)$ when considering t'' as derivative of t_6, with the latter formula not distinguishing be-

tween t_5 and t_6. Moreover, when focussing on t_5''', no formula ϕ' could have been found such that $t_5''' \not\models \phi' \rightleftharpoons t''$ as $\Phi_\vee(t'') \subsetneq \Phi_\vee(t_5''')$. ■

The last example shows that, in the general format $\langle a\rangle_{1-(\Delta_{2,a}(t')+p)} \bigvee_{j\in J} \phi'_{2,j}$ for the $\mathrm{PML}_\vee$ distinguishing formula mentioned at the beginning of this subsection, the set J only contains any derivative of the second node different from t' to which the two distributions assign two *different* probabilities. No derivative of the two original nodes having the same probability in both distributions is taken into account even if its $\Phi_\vee$-set is minimal – because it might be useless for discriminating purposes – nor is it included in J – because there might be no formula satisfied by this node when viewed as a derivative of the second node, which is not satisfied by t'. Furthermore, the value p is the probability that the second node reaches the excluded derivatives that do *not* satisfy $\bigvee_{j\in J} \phi'_{2,j}$; note that the first node reaches those derivatives with the same probability p.

We present two additional examples illustrating some technicalities of Def. 3. The former example shows the usefulness of the operator $\dot\vee$ and of the function *hplb* for selecting the right t' on the basis of the minimality of its $\Phi_\vee$-set among the derivatives of the first node to which the first distribution assigns a probability greater than the second one. The latter example emphasizes the role played, for the same purpose as before, by formulas occurring in a $\Phi_\vee$-set whose number of nested diamonds is not maximal.

Example 6. For the nodes t_9 and t_{10} in Fig. 1(e), we have:

$$\Phi_\vee(t_9) = \{\langle a\rangle_1,\ \langle a\rangle_{0.5}\langle b\rangle_1,\ \langle a\rangle_{0.5}\langle c\rangle_1\}$$
$$\Phi_\vee(t_{10}) = \{\langle a\rangle_1,\ \langle a\rangle_{0.5}\langle b\rangle_1,\ \langle a\rangle_{0.5}\langle c\rangle_1,\ \langle a\rangle_{0.6}\,(\langle b\rangle_1 \vee \langle c\rangle_1)\}$$

Starting from the two identically labeled transitions $t_9 \xrightarrow{a} \Delta_{9,a}$ and $t_{10} \xrightarrow{a} \Delta_{10,a}$ where $\Delta_{9,a}(t') = \Delta_{9,a}(t'') = 0.5$, $\Delta_{10,a}(t') = \Delta_{10,a}(t'') = 0.4$, $\Delta_{10,a}(t_{10}''') = \Delta_{10,a}(t_{10}'''') = 0.1$, and $\Delta_{9,a}(t_{10}''') = \Delta_{9,a}(t_{10}'''') = 0$, we have:

$$\Phi_\vee(t') = \{\langle b\rangle_1,\ \langle c\rangle_1\} \quad \Phi_\vee(t'') = \emptyset \quad \Phi_\vee(t_{10}''') = \{\langle b\rangle_1\} \quad \Phi_\vee(t_{10}'''') = \{\langle c\rangle_1\}$$

If we focus on t'' because $\Delta_{9,a}(t'') > \Delta_{10,a}(t'')$ and its $\Phi_\vee$-set is minimal, then $t'' \not\models \langle b\rangle_1 \rightleftharpoons t'$ with $\langle b\rangle_1 \in \Phi_\vee(t') \setminus \Phi_\vee(t'')$, $t'' \not\models \langle b\rangle_1 \rightleftharpoons t_{10}'''$ with $\langle b\rangle_1 \in \Phi_\vee(t_{10}''') \setminus \Phi_\vee(t'')$, and $t'' \not\models \langle c\rangle_1 \rightleftharpoons t_{10}''''$ with $\langle c\rangle_1 \in \Phi_\vee(t_{10}'''') \setminus \Phi_\vee(t'')$. Thus, $t_9 \not\models \langle a\rangle_{0.6}\,(\langle b\rangle_1 \vee \langle c\rangle_1) \rightleftharpoons t_{10}$ where the formula belongs to $\Phi_\vee(t_{10})$ and the value 0.6 decorating the a-diamond stems from $1 - \Delta_{10,a}(t'')$.

If $\vee$ were used in place of $\dot\vee$, then in $\Phi_\vee(t_{10})$ we would also have formulas like $\langle a\rangle_{0.5}\,(\langle b\rangle_1 \vee \langle b\rangle_1)$ and $\langle a\rangle_{0.5}\,(\langle c\rangle_1 \vee \langle c\rangle_1)$. These are useless in that logically equivalent to other formulas already in $\Phi_\vee(t_{10})$ in which disjunction does not occur and, most importantly, would apparently augment the size of $\Phi_\vee(t_{10})$, an inappropriate fact in the case that t_{10} were a derivative of some other node instead of being the root of a tree.

If *hplb* were not used, then in $\Phi_\vee(t_{10})$ we would also have formulas like $\langle a\rangle_{0.1}\langle b\rangle_1$, $\langle a\rangle_{0.4}\langle b\rangle_1$, $\langle a\rangle_{0.1}\langle c\rangle_1$, and $\langle a\rangle_{0.4}\langle c\rangle_1$, in which the probabilistic lower bounds of the a-diamonds are not maximal with respect to the satisfiability of formulas of that form by t_{10}; those with maximal probabilistic lower bounds associated with a-diamonds are $\langle a\rangle_{0.5}\langle b\rangle_1$ and $\langle a\rangle_{0.5}\langle c\rangle_1$, which already belong to $\Phi_\vee(t_{10})$. In the case that t_9 and t_{10} were derivatives of two nodes under

comparison instead of being the roots of two trees, the presence of those additional formulas in $\Phi_\vee(t_{10})$ may lead to focus on t_{10} instead of t_9 – for reasons that will be clear in Ex. 8 – thereby producing no distinguishing formula. ■

Example 7. For the nodes t_{11}, t_{12}, t_{13} in Fig. 1(f), we have:
$\Phi_\vee(t_{11}) = \{\langle a\rangle_1\}$ $\Phi_\vee(t_{12}) = \{\langle a\rangle_1, \langle a\rangle_1\langle b\rangle_1\}$ $\Phi_\vee(t_{13}) = \{\langle a\rangle_1, \langle a\rangle_{0.7}\langle b\rangle_1\}$
Let us view them as derivatives of other nodes, rather than roots of trees. The presence of formula $\langle a\rangle_1$ in $\Phi_\vee(t_{12})$ and $\Phi_\vee(t_{13})$ – although it has not the maximum number of nested diamonds in those two sets – ensures the minimality of $\Phi_\vee(t_{11})$ and hence that t_{11} is selected for building a distinguishing formula. If $\langle a\rangle_1$ were not in $\Phi_\vee(t_{12})$ and $\Phi_\vee(t_{13})$, then t_{12} and t_{13} could be selected, but no distinguishing formula satisfied by t_{11} could be obtained. ■

The criterion for selecting the right t' based on the minimality of its $\Phi_\vee$-set has to take into account a further aspect related to *formulas without disjunctions*. If two derivatives – with different probabilities in the two distributions – have the same formulas without disjunctions in their $\Phi_\vee$-sets, then a distinguishing formula for the two nodes will have disjunctions in it (see Exs. 4 and 6). If the formulas without disjunctions are different between the two $\Phi_\vee$-sets, then one of them will tell the two derivatives apart (see Ex. 3).

A particular instance of the second case is the one in which for each formula without disjunctions in one of the two $\Phi_\vee$-sets there is a variant in the other $\Phi_\vee$-set – i.e., a formula without disjunctions that has the same format but may differ for the values of some probabilistic lower bounds – and vice versa. In this event, *regardless of the minimality* of the $\Phi_\vee$-sets, it has to be selected the derivative such that (i) for each formula without disjunctions in its $\Phi_\vee$-set there exists a variant in the $\Phi_\vee$-set of the other derivative such that the probabilistic lower bounds in the former formula are $\leq$ than the corresponding bounds in the latter formula and (ii) at least one probabilistic lower bound in a formula without disjunctions in the $\Phi_\vee$-set of the selected derivative is $<$ than the corresponding bound in the corresponding variant in the $\Phi_\vee$-set of the other derivative. We say that the $\Phi_\vee$-set of the selected derivative is a $(\leq, <)$-*variant* of the $\Phi_\vee$-set of the other derivative.

Example 8. Let us view the nodes t_5 and t_6 in Fig. 1(c) as derivatives of other nodes, rather than roots of trees. Based on their $\Phi_\vee$-sets shown in Ex. 5, we should focus on t_6 because $\Phi_\vee(t_6)$ contains fewer formulas. However, by so doing, we would be unable to find a distinguishing formula in $\Phi_\vee(t_5)$ that is not satisfied by t_6. Indeed, if we look carefully at the formulas without disjunctions in $\Phi_\vee(t_5)$ and $\Phi_\vee(t_6)$, we note that they differ only for their probabilistic lower bounds: $\langle a\rangle_1 \in \Phi_\vee(t_6)$ is a variant of $\langle a\rangle_1 \in \Phi_\vee(t_5)$, $\langle a\rangle_{0.5}\langle b\rangle_1 \in \Phi_\vee(t_6)$ is a variant of $\langle a\rangle_{0.25}\langle b\rangle_1 \in \Phi_\vee(t_5)$, and $\langle a\rangle_{0.5}\langle c\rangle_1 \in \Phi_\vee(t_6)$ is a variant of $\langle a\rangle_{0.25}\langle c\rangle_1 \in \Phi_\vee(t_5)$. Therefore, we must focus on t_5 because $\Phi_\vee(t_5)$ contains formulas without disjunctions such as $\langle a\rangle_{0.25}\langle b\rangle_1$ and $\langle a\rangle_{0.25}\langle c\rangle_1$ having smaller bounds: $\Phi_\vee(t_5)$ is a $(\leq, <)$-variant of $\Phi_\vee(t_6)$.

Consider now the nodes t_9 and t_{10} in Fig. 1(e), whose $\Phi_\vee$-sets are shown in Ex. 6. If function *hplb* were not used and hence $\Phi_\vee(t_{10})$ also contained $\langle a\rangle_{0.1}\langle b\rangle_1$,

$\langle a\rangle_{0.4}\langle b\rangle_1$, $\langle a\rangle_{0.1}\langle c\rangle_1$, and $\langle a\rangle_{0.4}\langle c\rangle_1$, then the formulas without disjunctions in $\Phi_\vee(t_9)$ would no longer be equal to those in $\Phi_\vee(t_{10})$. More precisely, the formulas without disjunctions would be similar between the two sets, with those in $\Phi_\vee(t_{10})$ having smaller probabilistic lower bounds, so that we would erroneously focus on t_{10}. ■

Summing up, in the $\mathrm{PML}_\vee$ distinguishing formula $\langle a\rangle_{1-(\Delta_{2,a}(t')+p)} \bigvee_{j\in J} \phi'_{2,j}$, the steps for choosing the derivative t', on the basis of which each subformula $\phi'_{2,j}$ is then generated so that it is not satisfied by t' itself, are the following:

1. Consider only derivatives to which $\Delta_{1,a}$ assigns a probability greater than the one assigned by $\Delta_{2,a}$.
2. Within the previous set, eliminate all the derivatives whose $\Phi_\vee$-sets have $(\leq,<)$-variants.
3. Among the remaining derivatives, focus on one of those having a minimal $\Phi_\vee$-set.

Theorem 6. *Let $(T, A, \longrightarrow)$ be in RPT_f and $t_1, t_2 \in T$. Then $t_1 = t_2$ iff $t_1 \models \phi \iff t_2 \models \phi$ for all $\phi \in PML_\vee$. Moreover, if $t_1 \neq t_2$, then there exists $\phi \in PML_\vee$ distinguishing t_1 from t_2 such that $depth(\phi) \leq \max(height(t_1), height(t_2))$.*

4.4 $\mathrm{PML}_\wedge$ Characterizes $\sim_{\mathrm{PB}}$: A Direct Proof for Discrete Systems

By adapting the proof of Thm. 6 consistently with the proof of Thm. 4, we can also prove that $\mathrm{PML}_\wedge$ characterizes $\sim_{\mathrm{PB}}$ by working directly on *discrete* state spaces.

The idea is to obtain $t_1 \models \langle a\rangle_{\Delta_{1,a}(t')+p} \bigwedge_{j\in J} \phi'_{2,j} \not\models t_2$. For any node t of finite height, we define the set $\Phi_\wedge(t)$ of $\mathrm{PML}_\wedge$ formulas satisfied by t featuring, in addition to maximal probabilistic lower bounds and diamonds arising only from transitions of t as for $\Phi_\vee(t)$, conjunctions that (i) stem only from transitions departing from the *same node* in the support of a distribution reached by t and (ii) are preceded by a diamond decorated with the *sum* of the probabilities assigned by that distribution to that node and other nodes with the *same transitions* considered for that node. Given t having transitions of the form $t \xrightarrow{a_i} \Delta_i$ with $supp(\Delta_i) = \{t'_{i,j} \mid j \in J_i\}$ and $i \in I \neq \emptyset$, we let: $\Phi_\wedge(t) = \{\langle a_i\rangle_1 \mid i \in I\} \cup \bigcup_{i\in I} splb(\{\!|\, \langle a_i\rangle_{\Delta_i(t'_{i,j})} \bigwedge_{k\in K'} \phi'_{i,j,k} \mid \emptyset \neq K' \subseteq K_{i,j}, t'_{i,j} \in supp(\Delta_i), \phi'_{i,j,k} \in \Phi_\wedge(t'_{i,j}) \,|\!\})$ where $\{\!|$ and $|\!\}$ are multiset parentheses, $K_{i,j}$ is the index set for $\Phi_\wedge(t'_{i,j})$, and function *splb* merges all formulas possibly differring only for the probabilistic lower bound decorating their initial a_i-diamond by summing up those bounds (such formulas stem from different nodes in $supp(\Delta_i)$).

A good criterion for choosing t' occurring in the $\mathrm{PML}_\wedge$ distinguishing formula at the beginning of this subsection is the *maximality* of the $\Phi_\wedge$-set. Moreover, in that formula J only contains any derivative of the second node different from t' to which the two distributions assign two *different* probabilities, while p is the probability of reaching derivatives having the *same* probability in both distributions that *satisfy* $\bigwedge_{j\in J} \phi'_{2,j}$. Finally, when selecting t', we have to leave out all the derivatives whose $\Phi_\wedge$-sets have $(\leq,<)$-variants.

Theorem 7. *Let* $(T, A, \longrightarrow)$ *be in* RPT_f *and* $t_1, t_2 \in T$. *Then* $t_1 = t_2$ *iff* $t_1 \models \phi \iff t_2 \models \phi$ *for all* $\phi \in PML_\wedge$. *Moreover, if* $t_1 \neq t_2$, *then there exists* $\phi \in PML_\wedge$ *distinguishing* t_1 *from* t_2 *such that* $depth(\phi) \leq \max(height(t_1), height(t_2))$.

5 Conclusions

In this paper, we have studied modal logic characterizations of strong bisimilarity over reactive probabilistic processes. Starting from previous work by Larsen and Skou [12] (who provided a characterization based on a probabilistic extension of Hennessy-Milner logic) and by Desharnais, Edalat, and Panangaden [6] (who showed that negation is not necessary), we have proved that conjunction can be replaced by disjunction without having to reintroduce negation. Thus, in the reactive probabilistic setting, conjunction and disjunction are interchangeable to characterize (bi)simulation equivalence, while they are both necessary for simulation preorder [7]. As a side result, with our proof technique we have provided alternative proofs of the expressiveness of $\text{PML}_{\neg\wedge}$ and $\text{PML}_\wedge$.

The intuition behind our result is that from a conjunctive distinguishing formula it is often possible to derive a disjunctive one by suitably increasing some probabilistic lower bounds. On the model side, this corresponds to summing up the probabilities of reaching certain states that are in the support of a target distribution. In fact, a state of an RPLTS can be given a semantics as a *reactive probabilistic tree*, and hence it is characterized by the countable set of formulas (approximated by the $\Phi_\vee$-set) obtained by doing finite visits of the tree.

On the application side, the $\text{PML}_\vee$-based characterization of bisimilarity over reactive probabilistic processes may help to prove a conjecture in [4]. This work studies the discriminating power of three different testing equivalences respectively using reactive probabilistic tests, fully nondeterministic tests, and nondeterministic and probabilistic tests. Numerous examples lead to conjecture that testing equivalence based on nondeterministic and probabilistic tests may have the same discriminating power as bisimilarity. Given two $\sim_{\text{PB}}$-inequivalent reactive probabilistic processes, the idea of the tentative proof is to build a distinguishing nondeterministic and probabilistic test from a distinguishing $\text{PML}_\wedge$ formula. One of the main difficulties with carrying out such a proof, i.e., the fact that choices within tests fit well together with disjunction rather than conjunction, may be overcome by starting from a distinguishing $\text{PML}_\vee$ formula.

References

1. P. Aczel and N. Mendler. A final coalgebra theorem. In *Proc. of the 3rd Conf. on Category Theory and Computer Science (CTCS 1989)*, volume 389 of *LNCS*, pages 357–365. Springer, 1989.
2. C. Baier and M. Kwiatkowska. Domain equations for probabilistic processes. *Mathematical Structures in Computer Science*, 10:665–717, 2000.
3. M. Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114:299–315, 1993.

4. M. Bernardo, D. Sangiorgi, and V. Vignudelli. On the discriminating power of testing equivalences for reactive probabilistic systems: Results and open problems. In *Proc. of the 11th Int. Conf. on the Quantitative Evaluation of Systems (QEST 2014)*, volume 8657 of *LNCS*, pages 281–296. Springer, 2014.
5. Y. Deng and H. Wu. Modal characterisations of probabilistic and fuzzy bisimulations. In *Proc. of the 16th Int. Conf. on Formal Engineering Methods (ICFEM 2014)*, volume 8829 of *LNCS*, pages 123–138. Springer, 2014.
6. J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179:163–193, 2002.
7. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labelled Markov processes. *Information and Computation*, 184:160–200, 2003.
8. R.J. van Glabbeek. The linear time – branching time spectrum I. In *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.
9. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.
10. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–162, 1985.
11. B. Jacobs and A. Sokolova. Exemplaric expressivity of modal logics. *Journal of Logic and Computation*, 20:1041–1068, 2010.
12. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
13. P. Panangaden. Probabilistic bisimulation. In *Advanced Topics in Bisimulation and Coinduction*, pages 300–334. Cambridge University Press, 2011.
14. E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Theoretical Computer Science*, 221:271–293, 1999.
15. J. Worrell. On the final sequence of a finitary set functor. *Theoretical Computer Science*, 338:184–199, 2005.

Short Communications

Reversible Semantics in Session-based Concurrency*

Claudio Antares Mezzina[1] and Jorge A. Pérez[2]

[1] IMT School for Advanced Studies Lucca, Italy
[2] University of Groningen, The Netherlands

Abstract. Much research has studied foundations for correct and reliable communication-centric systems. A salient approach to correctness uses session types to enforce structured communications; a recent approach to reliability uses reversible actions as a way of reacting to unanticipated events or failures.
This note describes recent work that develops a simple observation: the machinery required to define monitored semantics can also support reversible protocols. We illustrate a process framework of session communication in which monitors support reversibility. A key novelty in our approach are session types with present and past, which allow us to streamline the semantics of reversible actions.

1 Introduction

The purpose of this short paper is to motivate and describe our ongoing work in reversible models of structured communications [8]. Framed within concurrency theory and process calculi approaches, we are interested in developing rich models of concurrent computation in which communicating processes follow structured interaction protocols (as described by *session types* [3]), and whose underlying operational semantics admits the possibility of reversing their actions. This integration of structured communication and reversibility should inform the design of sound programming abstractions for resilient communicating programs governed by casual consistent semantics.

Models of reversible computation and structured communications have received much attention (cf. [1,3]). Reversing computational steps is an appealing feature in different scenarios; for instance, in the case of a failure in a (concurrent) program or transaction, we might like to undo all steps leading to the failure, so

* Partially supported by EU COST Actions IC1201 (Behavioral Types for Reliable Large-Scale Software Systems) and IC1405 (Reversible Computation - Extending Horizons of Computing).

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 221–226 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

as to return to last known stable state of the system. Indeed, good examples of how reversibility can be used to model transactional models are [2,6]. The design of reversible semantics for models of concurrency is a delicate issue, for we would like to undo computational steps in a *causally consistent* fashion: a step should be undone only if all its causes (if any) have been already undone. In this way, reversibility in a causal consistent model leads to a system state that could have been reached by performing forward steps only.

The key observation that underlies our work is that the design of casually consistent operational semantics for concurrent processes can take advantage of the structured protocols that typically govern their behavior. As session types abstract sequences of communication actions (protocols), they appear as a natural choice for recording the forward and backward actions of interacting processes.

In recent work, we have started to formalize the integration of reversibility and session-based concurrency [8]. In this note, we illustrate the model in [8] by means of a simple example that contains the main ingredients of our approach, namely an operational semantics for untyped processes which is instrumented by *monitors* that contain protocols specified as session types. In order to support both forward and backward steps, we consider session types that describe both *past* and *present* protocol states.

2 Reversible Sessions, By Example

Our proposal builds upon the approach of models of concurrency such as the π-calculus. As such, main ingredients in our approach are *configurations*, *processes*, and *(protocol) types*, whose syntax is given in Figure 1. We assume a language of the expressions $e, e', \ldots$ that includes basic values, variables, and functions on them. The syntax of configurations includes the empty configuration $\mathbf{0}$, name restriction $\nu n.M$, parallel composition $M \parallel N$, but also *running processes* and *monitors*:

- A running process $\langle P \cdot \sigma \cdot \widetilde{u} \rangle_{\widetilde{s}}$ is univocally identified by $\widetilde{s}$, the list of session endpoints occurring in P. The local store σ is a list of pairs of the form $\{x, \widetilde{v}\}$ (i.e., a set of mappings from variables to values); the list $\widetilde{u}$ collects the subjects of actions already performed by P.
- Given a session name s, a monitor $s\lfloor H \cdot \widetilde{e} \rfloor$ contains a protocol (session) type H that describes the structured behavior associated to s (see below) and a list $\widetilde{e}$ containing all the expressions (including variables) used by the process.

Intuitively, the list $\widetilde{u}$ in the running process and the list $\widetilde{e}$ in the monitor will be used to record previously performed actions and reconstruct the process structure accordingly.

The design of the operational semantics for our model is inspired by the approach of [5], in which session types are used as *monitors* that enable communication actions: a synchronization can only occur if the process actions correspond to the intended protocols given by the monitor types. After synchronization, portions of both processes and monitor types are consumed. Our approach consists

$$
\begin{array}{llll}
(\textit{configurations}) & M, N ::= & \mathbf{0} \mid \langle P \cdot \sigma \cdot \widetilde{u} \rangle_{\widetilde{s}} \mid s\lfloor H \cdot \widetilde{e} \rfloor \mid \nu n.M \mid M \parallel N & \\
(\textit{processes}) & P, Q ::= & u(x : S).P \mid u\langle x : S \rangle.P \mid k\langle e \rangle.P \mid k(x).P \mid \nu a.P \mid \mathbf{0} & \\
(\textit{actions}) & \alpha, \beta ::= & !U \mid ?U & (\textit{protocol types}) \quad S, T ::= \quad \mathtt{end} \mid \alpha.S \\
(\textit{history types}) & H, K ::= & \hat{} S \mid S \hat{} \mid \alpha_1.\cdots.\alpha_n.\hat{} S &
\end{array}
$$

Fig. 1. Syntax of Configurations, Processes, and Types.

in keeping, rather than consuming, these monitor types. For this to work, we need to distinguish the part of the protocol that has been already executed (its past), from the protocol that still needs to execute (its present). We thus introduce session types with present and past (H in the syntax): the type $\alpha_1.\cdots.\alpha_n.\hat{} S$ says that actions $\alpha_1, \cdots, \alpha_n$ are past protocol actions, whereas actions in protocol S are yet to be executed. That is, the cursor $\hat{}$ in history types helps us to distinguish the past from the present. Each action α_i corresponds to the input or output of a value; we use U to range over the types of these values (e.g., **int**, **str**, etc.).

We illustrate our approach by means of a simple business protocol example [4]: a slightly modified version of the *two buyers protocol*. It involves three participants: a Buyer, a Seller, and a Buyer's Friend. Buyer is willing to buy a book, and sends to Seller the title of the book he is interested in. Seller replies with the price of the book, and awaits for final information (e.g., *shipping address* and *order confirmation*) from Buyer, before providing a *delivery date*. Once Buyer receives the price, he realizes that he needs a loan from Friend in order to finalize the purchase. To this aim, Buyer contacts Friend and then the transaction is finalized. The set of interactions of Buyer with Seller and Friend are prescribed by the following session types:

$$
\begin{array}{ll}
S_a : ?\mathbf{str}.!\mathbf{int}.?\mathbf{str}.?\mathbf{int}.!\mathbf{cal}.\mathtt{end} & S_b : ?\mathbf{int}.!\mathbf{int}.\mathtt{end} \\
T_a : !\mathbf{str}.?\mathbf{int}.!\mathbf{str}.!\mathbf{int}.?\mathbf{cal}.\mathtt{end} & T_b : !\mathbf{int}.?\mathbf{int}.\mathtt{end}
\end{array}
$$

Above, S_a describes the interaction between Buyer and Seller from Seller's perspective; type T_a is its *dual* and describes the protocol from Buyer's perspective. In session types, duality is essential to (statically) ensure action compatibility between partners (and therefore, to guarantee absence of communication errors). Types T_b and S_b describe the interaction between Buyer and Friend, from each perspective.

Having defined the interaction protocols using types, we proceed to examine some possible process implementations for Buyer, Seller, and Friend. The behavior

of Buyer may be specified by the following process:

$$\textsc{Buyer} \triangleq a\langle z : T_a\rangle.z\langle\text{“dune”}\rangle.z(prc).\\ b\langle w : T_b\rangle.w\langle \texttt{loan}(prc)\rangle.w(cash).z\langle addr\rangle.z\langle cash\rangle.z(date).\mathbf{0}$$

The implementation for Buyer involves the creation of two interleaved sessions: the first one is established with the prefix $a\langle z : T_a\rangle$, which explicitly mentions the session protocol to be executed with the implementation of Seller; the second session is established with the implementation of Friend through the prefix $b\langle w : T_b\rangle$. Process implementations for Seller and Friend can be specified by the following processes:

$$\textsc{Seller} \triangleq a(z : S_a).z(title).z\langle\texttt{quote}(title)\rangle.z(addr).z(paymnt).z\langle\texttt{date}(addr)\rangle.\mathbf{0}$$
$$\textsc{Friend} \triangleq b(w : S_b).w(amount).w\langle loan\rangle.\mathbf{0}$$

Note that functions `loan`(), `quote`() and `date`() are used to calculate the amount of money to be borrowed, the book price and the delivery date, respectively. The overall system specification is then given by the parallel composition of configurations containing the three processes (in what follows, ϵ denotes the empty list):

$$\textsc{System} \triangleq \left\langle \textsc{Buyer}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon \parallel \left\langle \textsc{Seller}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon$$

In the following, we will indicate with $\textsc{Buyer}_i$ (resp. $\textsc{Seller}_i$ and $\textsc{Friend}_i$) the process Buyer after performing its i-th action. We will do the same with types.

The operational semantics that we have defined in [8] is based on a reduction relation with both forward and backward steps, denoted $\twoheadrightarrow$ and $\rightsquigarrow$, respectively. The first forward reduction of System is establishing a session between Buyer and Seller, using the fact that T_a and S_a are dual types. We have:

$$\textsc{System} \twoheadrightarrow (\nu s,\overline{s}).\Big(\left\langle \textsc{Buyer}_1\cdot\{z,s\}\cdot a\right\rangle_s \parallel s\lfloor\frown T_a\cdot z\rfloor \parallel \\ \left\langle \textsc{Seller}_1\cdot\{z,\overline{s}\}\cdot a\right\rangle_{\overline{s}} \parallel \overline{s}\lfloor\frown S_a\cdot z\rfloor \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon\Big) \tag{1}$$

As we can see, once a session is established two monitors are created, one per endpoint; their task is to discipline the behavior of the process holding the endpoint. For example, the behavior of Buyer in session s has to obey type S_a. Buyer then sends (according to S_a) to Seller the request for the book, and the entire system evolves as:

$$\twoheadrightarrow(\nu s,\overline{s}).\Big(\left\langle \textsc{Buyer}_2\cdot(\{z,s\})\cdot a,z\right\rangle_s \parallel s\lfloor!\texttt{str}\frown T_{a_1}\cdot z,\text{“dune”}\rfloor \parallel \\ \left\langle \textsc{Seller}_2\cdot(\{z,\overline{s}\},\{title,\text{“dune”}\})\cdot a,z\right\rangle_{\overline{s}} \parallel \\ \overline{s}\lfloor?\texttt{str}\frown S_{a_1}\cdot z,title\rfloor \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon\Big) = M \tag{2}$$

As effect of the communication, both types register the action and move forward. Another effect is that the information needed to restore back the consumed prefixes is stored into the running configurations and the related monitors. Communication in (2) can be reverted by moving backward the monitor types, by restoring the prefixes and deleting the read value from the receiver store, that is:

$$M \rightsquigarrow (\nu s, \overline{s}).\Big(\big\langle z\langle \text{“dune”}\rangle.\textsc{Buyer}_2 \cdot (\{z,s\} \cdot a\big\rangle_s \parallel s\lfloor \frown !\texttt{str}.T_{a_1} \cdot z\rfloor \parallel$$
$$\big\langle z(\mathit{title}).\textsc{Seller}_2 \cdot \{z,\overline{s}\} \cdot a\big\rangle_{\overline{s}} \parallel \overline{s}\lfloor \frown ?\texttt{str}.S_{a_1} \cdot z\rfloor \parallel \big\langle \textsc{Friend} \cdot \epsilon \cdot \epsilon\big\rangle_{\epsilon}\Big) \tag{3}$$

We can easily check that the configurations in (3) and (1) are equivalent. From M in (2) the interaction between Buyer and Seller goes on, and the system arrives to a point in which Buyer establishes a new session with Friend:

$$M \twoheadrightarrow^* (\nu s, \overline{s}, r, \overline{r}).\Big(\big\langle \textsc{Buyer}_4 \cdot (\{z,s\},\{w,r\}) \cdot \widetilde{u}_1, b\big\rangle_{s,r} \parallel$$
$$r\lfloor \frown T_b \cdot b\rfloor \parallel s\lfloor T'_a \frown T_{a_3} \cdot z, \text{“dune”}, \mathit{prc}, w\rfloor \parallel$$
$$\big\langle \textsc{Seller}_3 \cdot (\{z,\overline{s}\},\{\mathit{title}, \text{“dune”}\}) \cdot a, z, z\big\rangle_{\overline{s}} \parallel$$
$$\overline{s}\lfloor S'_a \frown S_{a_3} \cdot z, \mathit{title}, \texttt{quote}(\mathit{title})\rfloor \parallel \big\langle \textsc{Friend}_1 \cdot \{w,\overline{r}\} \cdot b\big\rangle_{\overline{r}} \parallel \overline{r}\lfloor \frown S_b \cdot w\rfloor\Big) \tag{4}$$

As (4) shows, the running process for Buyer is present in two sessions: one with Seller and another one with Friend, and has two associated monitors, identified by endpoints s, r. The list of subjects stored into the running process allows us to reverse communications (possibly in different sessions) and session establishments in the order in which they were performed, thus respecting causality of actions. In this way, Buyer cannot undo a communication with Seller while the session with Friend is still established.

3 Future Work

We have described recent work on the integration of reversible semantics and session-based concurrency [8]. It represents a fresh approach with respect to previous approaches [9]. Several directions deserve further investigation:

- *Richer (typed) languages.* The process model in [8] is admittedly simple; to model and reason about interesting examples we need support for constructs such as labeled choices. Also, process specifications do not specify reversible actions; this is the role of monitors, history types, and other mechanisms. Since reversibility is independent from specifications, rich types are needed to support controlled forms of reversibility. In recent work we propose alternatives to these challenges [7].

- *Multiparty session communications.* The model in [8] concerns *binary* session types, which codify interaction between exactly two partners. Generalizing our approach to *multiparty* session types [4] should require a finer, coordinated representation of reversible actions, as protocol exchanges may involve more than two participants.
- *Dedicated reasoning techniques.* Session types induce a "simpler" model of concurrency in which reversibility is a better behaved phenomenon. It remains to be seen to what extent such a setting enables the development of tractable reasoning techniques (e.g., axiomatizations, behavioral equivalences, and proof systems).

References

1. Danos, V., Krivine, J.: Reversible communicating systems. In: Proc. of CONCUR 2004. pp. 292–307. LNCS, Springer (2004)
2. Danos, V., Krivine, J.: Transactions in RCCS. In: CONCUR 2005. LNCS, vol. 3653, pp. 398–412 (2005)
3. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP'98. LNCS, vol. 1381, pp. 122–138. Springer (1998)
4. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL 2008. pp. 273–284. ACM (2008)
5. Kouzapas, D., Yoshida, N., Honda, K.: On asynchronous session semantics. In: Proc. of FMOODS 2011 and FORTE 2011. LNCS, vol. 6722, pp. 228–243. Springer (2011)
6. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Concurrent flexible reversibility. In: ESOP 2013. LNCS, vol. 7792, pp. 370–390 (2013)
7. Mezzina, C.A., Pérez, J.A.: Reversibility in session-based concurrency: A fresh look (2016), draft available in `http://www.jperez.nl`
8. Mezzina, C.A., Pérez, J.A.: Reversible sessions using monitors. In: Proc. of PLACES 2016. EPTCS, vol. 211, pp. 56–64 (2016), `http://dx.doi.org/10.4204/EPTCS.211.6`
9. Tiezzi, F., Yoshida, N.: Reversible session-based pi-calculus. J. Log. Algebr. Meth. Program. 84(5), 684–707 (2015)

Towards A Practical Model of Reactive Communication-Centric Software*

Jaime Arias[1], Mauricio Cano[2], and Jorge A. Pérez[2]

[1] University of Bordeaux, CNRS LaBRI UMR, INRIA, France
[2] University of Groningen, The Netherlands

Abstract. Many distributed software systems are *communication-centric*: they are composed of heterogeneous software artifacts that interact following precise communication structures (*protocols*). One much-studied approach to system analysis equips process calculi with *behavioral types* (such as *session types*) so to abstract protocols and verify interacting programs. Unfortunately, existing behaviorally typed frameworks do not adequately support *reactive behavior*, an increasingly relevant feature in protocols. To address this shortcoming, We have been exploring how the *synchronous programming* paradigm can uniformly support the formal analysis of reactive, communication-centric programs. In this short communication, we motivate our approach and report on ongoing developments.

1 Introduction

In this short note, we describe our ongoing work on a *reactive* programming model for *communication-centric* software systems. While most previous work relies on models based on the π-calculus [14], we are developing practical support for communication-centric software systems using ReactiveML [13], a *synchronous* programming language with functional and reactive features, and which relies on solid formal foundations.

In communication-centric software systems, collections of heterogeneous software artifacts usually follow well-defined communication structures, or *protocols*. Ensuring that programs conform to these protocols is key to certify system correctness. One much-studied approach to the analysis of communicating programs uses *behavioral types* [12], a type-based verification technique that captures complex communication structures while enforcing resource-usage policies. *Session*

* Research partially supported by EU COST Action IC1201 (Behavioral Types for Reliable Large-Scale Software Systems) and CNRS PICS project 07313 (SuCCeSS).

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 227–233 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

types [11] are a class of behavioral types that organize protocols as *sessions* between two or more participants; a session type describes the contribution of each partner to the protocol. First formulated as a type theory for the π-calculus, session-based concurrency has been implemented as communication libraries for mainstream languages, such as OCaml [15] and Scala [16].

One shortcoming of existing implementations is that they are based on overly rigid programming models. In particular, current practical support for communication-centric software systems does not explicitly consider *reactive behavior* in communicating programs. This is a crucial feature, especially as autonomous agents can now engage into protocols in our behalf (e.g., financial transactions). In fact, reactive behavior is central in realistic implementations of protocols with, e.g., exception handling, dynamic reconfiguration, and time. While these features can be represented in languages based on the π-calculus (cf. e.g., [6,3]), resulting models are often difficult or unnatural to reason about. Session types themselves focus on representing communication structures and thus abstract away from aspects related to reactivity. As protocols in emerging applications are increasingly subject to external stimuli/events (typically hard to predict), developing programming support that uniformly integrates structured communications and flexible forms of reactive behavior appears as a pressing need.

To our knowledge, the amalgamation of reactive behavior into models of structured communications has been little explored by previous works (see, e.g., [9]). Our efforts have been triggered by our declarative interpretation of session-based concurrency [5]. Our current work goes beyond the interpretation in [5] so to consider reactive and declarative behavior from a programming languages perspective. To this end, we have developed an implementation of sessions in ReactiveML [13], supported by a formal translation of session processes as ReactiveML programs. Based on our preliminary results, we believe that models of reactive programming improve previous works by offering a uniform basis for expressing and reasoning about different kinds of constructs.

2 Session Types

Session types offer a type-based methodology to the validation of communicating programs [11]. Structured dialogues (protocols) between interacting parties are represented as *sessions*; sequences of interactions along each channel in the program/process are then abstracted as types, which can be used to (statically) verify whether a program conforms to its intended protocols. Key properties are *fidelity* (programs respect prescribed protocols) and *communication safety* (programs do not have errors, e.g., communication mismatches). The syntax of

(binary) session types T, S is as follows:

$!T.S$	Output a value of type T, continue as type/protocol S
$?T.S$	Receive a value of type T, continue as type/protocol S
$\&\{l_i : T_i\}_{i \in I}$	External choice among labeled types/protocols T_i (branching)
$\oplus\{l_i : T_i\}_{i \in I}$	Internal choice of a labeled type/protocol T_j, with $j \in I$ (selection)
$\mu X.T$	Recursive protocol/type (with type variable X)
end	Terminated protocol

In session-based concurrency, the notion of *duality* is key to ensure communication safety. Intuitively, duality relates session types with opposite behaviors: e.g., the dual of input is output, and vice versa; branching is the dual of selection, and vice versa.

We illustrate session types using a traditional example in the literature: the *Buyer-Seller-Shipper protocol*, which can be informally described as follows:

1. Buyer requests an item from Seller.
2. Seller replies back asking for Buyer's unique address.
3. Buyer sends his address to Seller, confirming the order.
4. Seller forwards Buyer's address to Shipper.
5. Shipper sends to Buyer the estimated delivery time.
6. Buyer confirms to Shipper his availability for receiving the item.

We may formalize this protocol using the following session types:

$$BuySell = \texttt{!item}.\texttt{?confirmation}.\texttt{!address}.\mathbf{end} \quad SellShip = \texttt{!address}.\mathbf{end}$$
$$ShipBuy = \texttt{!ETA}.\&\{yes : !ok.\mathbf{end}, no : !bye.\mathbf{end}\}$$

where `item`, `confirmation`, `address`, and `ETA` denote basic types. Type $BuySell$ describes interactions between Buyer and Seller from Buyer's perspective. Similarly, $SellShip$ describes an interaction from Seller's perspective, and $ShipBuy$ takes the standpoint of Shipper in communications. Complementary types can be obtained using duality. These three sessions take place in order, as in the informal description above.

3 A Reactive Approach to Communication-Centric Systems

The protocol presented before is well-suited for deployment using traditional technologies (e.g., web services). However, it does not consider the possibility of changes at runtime due to unexpected circumstances or external events. Moreover, the protocol is not suited to emerging scenarios in which protocol partners are deployed in, e.g., mobile devices with limited computational power and availability. For instance, it is easy to imagine Shipper being implemented by a drone with communication capabilities.

To address these shortcomings of protocol descriptions in session-based concurrency, we propose to use reactive behavior, as present in *synchronous reactive*

programming. In this context, we can envision a reactive variant of the Buyer-Seller-Shipper protocol, in which Shipper is a drone, and Buyer communicates from a mobile phone. In this variant of the protocol, the first six steps are as before; after Steps 1–6, an *event* from Buyer to Seller triggers the following protocol:

R1. Buyer adds an item to his recently completed order.
R2. Seller replies back confirming the modified order.
R3. Seller forwards the modified order to Shipper.
R4. Shipper replies back in one of the following ways:
    a) Shipper returns back to the store, picks up the new item, and confirms to Buyer the previously given estimated delivery time, or
    b) Shipper continues with the original order, and informs Buyer that the second item will be delivered separately.

That is, in the reactive Buyer-Seller-Shipper protocol some of the exchanges are "standard" or "default" (cf. Steps 1-6); there are also other exchanges that are executed as a reaction to some event or external circumstance (cf. Steps R1-R4). In the latter steps, the external event concerns the request by Buyer of modifying his order; other conceivable conditions include, e.g., drone malfunctioning and wrong/delayed package deliveries. These extra exchanges also constitute structured protocols, amenable to specification and validation using sessions; however, their occurrence can be very hard to predict.

Synchronous Reactive Programming and ReactiveML. Synchronous Reactive Programming (SRP) is an event-based model of computation, optimized for programming reactive systems [1]. Synchronous languages are based on the *hypothesis of perfect synchrony*: reactive programs respond *instantaneously* and produce their outputs synchronously with their input. A synchronous program is meant to deterministically react to events coming from the environment: in essence, it evolves through an infinite sequence of successive reactions indexed by a global logical clock. During a reaction, each system component computes new output values based on the input values and its internal state; the communication of all events between components occurs synchronously during each reaction. Reactions are required to converge and computations are entirely performed before the current execution instant ends and the next one begins. This notion of time enables SRP programs to have an *order* in the events of the system, which makes it possible to reason about some time-related properties [8,10].

ReactiveML is an SRP-based extension to the OCaml programming language [13], based on the reactive model presented in [4]. This model allows unbounded time response from processes and avoids causality issues that can occur in other approaches to SRP, such as the one used by ESTEREL [2]. ReactiveML extends OCaml with the notion of *processes*, which are state machines whose behavior can be executed through several logical instants. Processes are considered the reactive counterpart of OCaml functions, which are considered as instantaneous in ReactiveML.

In ReactiveML, synchronization is based on *signals*: events that occur in one logical instant. Signals can trigger reactions in processes, to be executed instantaneously or in the next time unit. Signals can carry values and can be emitted from different processes in the same logical instant. There are three basic ReactiveML constructs:

`emit s v`	emits signal `s` with value `v`.
`await one s <e> in P`	awaits a value along signal `s` that is pattern-matched to expression `<e>`. Process `P` is executed in the next instant.
`signal s in P`	declares signal `s` and bounds it to continuation `P`.

Our Current Work: Structured Communications in SRP. Models of communication-centric systems (such as session types) usually rely on directed exchanges along named channels. However, in SRP there is no native notion of channels: as we have seen, signals are the main synchronization unit in ReactiveML. To deal with this discrepancy, a key idea in our work is "simulating" channels using signals. To this end, and since we would like to represent protocols respecting linearity, we follow the representation of session channels in [7], which uses a continuation-passing style. This means that for each interaction within a communication structure a new channel is created.

We describe our ReactiveML implementation for Seller in the reactive Buyer-Seller-Shipper protocol. Recall that Seller is involved in two sessions: he first communicates with Client, then interactions with Shipper occur. In the code below we assume that all sessions have been already initiated; these are noted `cb` for Buyer and `cs` for Seller.

```
let process seller conf =
  await one cb (item,y) in signal c1 in
  emit y (conf,c1);pause;
  await one c1 (addr,u) in signal c2 in
  emit cs (addr,c2);pause
```

The code declares `seller` as a process in ReactiveML (a non-instantaneous function); we describe its body. The first line awaits a signal `cb`, which carries a pair of elements: a value and a reference to the signal where further interactions will occur (i.e., `y`). Then, a signal `c1`, where the next interaction will occur, is declared. Subsequently, a pair (containing a message `conf` and a reference to signal `c1`) is emitted over signal `y`; no further actions are taken in this time unit. Once the message is received by Buyer, `seller` awaits Buyer's address. At this point, the first session has finished and communication with Shipper begins. In the last line, Buyer's address is sent to Shipper.

Notice that once `seller` is finished, so is any communication from Seller. But in the reactive protocol, Seller must await possible further actions from either Buyer or Shipper. To implement this key feature, we extend the previous code with the following line of code: `await one g (v,e) in P`. This new line puts `seller` to "sleep" until an event/signal `g` from either Buyer or Seller triggers a new

reaction `P` from it. Note that this signal for *interrupts* or events should be known to every party in the communication. The idea is then that the continuation process `P` should decide which course of action to take depending on the value carried by `g`. In our reactive protocol, process `P` could implement Steps R1-R4, following the implementation scheme of `seller`.

4 Concluding Remarks

We have described our ongoing implementation of essential features of session-based concurrency in ReactiveML, a reactive functional programming language. Our implementation uses ReactiveML processes to handle usual session protocols (send, receive, select, and branch constructs). We believe that our approach is already on a par with other session implementations (such as [15]) with substantial room for improvement, due to the reactive behavior supported by ReactiveML. We expect our research to enable the practical use of session-based concurrency into emerging application scenarios, such as, e.g., Collective Adaptive Systems (CAS).

References

1. A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.
2. G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.
3. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *Proc. of CONCUR'14*, volume 8704, pages 419–434. Springer, 2014.
4. F. Boussinot and R. de Simone. The SL synchronous language. *IEEE Trans. Software Eng.*, 22(4):256–266, 1996.
5. M. Cano, C. Rueda, H. A. López, and J. A. Pérez. Declarative interpretations of session-based concurrency. In *Proc. of PPDP'15*, pages 67–78. ACM, 2015.
6. M. Carbone. Session-based choreography with exceptions. *Electr. Notes Theor. Comput. Sci.*, 241:35–55, 2009.
7. O. Dardha, E. Giachino, and D. Sangiorgi. Session types revisited. In *Proc. of PPDP'12*, pages 139–150, 2012.
8. R. de Simone, J. Talpin, and D. Potop-Butucaru. The synchronous hypothesis and synchronous languages. In *Embedded Systems Handbook.* CRC Press, 2005.
9. X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *Theor. Comput. Sci.*, 328(1-2):19–37, 2004.
10. A. Gamati. *Designing Embedded Systems with the SIGNAL Programming Language: Synchronous, Reactive Specification.* Springer, 1st edition, 2009.
11. K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *Proc. of ESOP'98*, volume 1381, pages 122–138. Springer, 1998.
12. H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P.-M. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, Apr. 2016.

13. L. Mandel and M. Pouzet. ReactiveML: a reactive extension to ML. In *Proc. of PPDP'05*, pages 82–93. ACM, 2005.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
15. L. Padovani. FuSe - A simple library implementation of binary sessions. URL: `http://www.di.unito.it/~padovani/Software/FuSe/FuSe.html`.
16. A. Scalas and N. Yoshida. Lightweight session programming in scala. In *ECOOP 2016*, LIPIcs. Dagstuhl, 2016.

Minimal and Reduced Reversible Automata*
(Extended Abstract)

Giovanna J. Lavado, Giovanni Pighizzini, and Luca Prigioniero

Dipartimento di Informatica, Università degli Studi di Milano, Italy
{lavado,pighizzini}@di.unimi.it, luca.prigioniero@studenti.unimi.it

Abstract. A condition characterizing the class of regular languages which have several nonisomorphic minimal reversible automata is presented. The condition concerns the structure of the minimum automaton accepting the language under consideration.
It is also observed that there exist reduced reversible automata which are not minimal, in the sense that all the automata obtained by merging some of their equivalent states are irreversible. Furthermore, it is proved that if the minimum deterministic automaton accepting a reversible language contains a loop in the "irreversible part" then it is always possible to construct infinitely many reduced reversible automata accepting such a language.

1 Introduction

A device is said to be *reversible* when each configuration has exactly one predecessor and one successor, thus implying that there is no loss of information during the computation. On the other hand, as observed by Landauer, logical irreversibility is associated with physical irreversibility and implies a certain amount of heat generation [7]. In order to avoid such a power dissipation and, hence, to reduce the overall power consumption of computational devices, the possibility of realizing reversible machines looks appealing.

A lot of work has been done to study reversibility in different computational devices. Just to give a few examples, in the case of general devices as Turing machines Bennet proved that each machine can be simulated by a reversible one [2], while Lange, McKenzie, and Tapp proved that each deterministic machine can be simulated by a reversible machine which uses the same amount of space [8]. As a corollary, in the case of a constant amount of space, this implies that

* Paper presented at the *18th International Conference on Descriptional Complexity of Formal Systems*, DCFS 2016. In LNCS 9777, pp. 168-179, 2016. DOI: 10.1007/978-3-319-41114-9_13

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 234–239 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

each regular language is accepted by a *reversible two-way deterministic finite automaton*. Actually, this result was already proved by Kondacs and Watrous [4].

However, in the case of *one-way automata*, the situation is different.[1] In fact, as shown by Pin, the regular language a^*b^* cannot be accepted by any reversible automaton [10]. So the class of languages accepted by reversible automata is a proper subclass of the class of regular languages. Actually, there are some different notions of reversible automata in literature. In 1982, Angluin introduced reversible automata in algorithmic learning theory, considering devices having only one initial and only one final state [1]. On the other hand, the devices considered in [10], besides a set of final states, can have multiple initial states, hence they can take a nondeterministic decision at the beginning of the computation. An extension which allows one to consider nondeterministic transitions, without changing the class of accepted languages, has been considered by Lombardy [9], introducing and investigating *quasi reversible automata*. Classical automata, namely automata with a single initial state and a set of final states, have been considered in the works by Holzer, Jakobi, and Kutrib [5,3,6]. In particular, in [3] the authors obtained a characterization of regular languages which are accepted by reversible automata. This characterization is given in terms of the structure of the minimum deterministic automaton. Furthermore, they provide an algorithm that, in the case the language is acceptable by a reversible automaton, allows one to transform the minimum automaton into an equivalent reversible automaton, which in the worst case is exponentially larger than the given minimum automaton. In spite of that, the resulting automaton is minimal, namely there are no reversible automata accepting the same language with a smaller number of states. However, the minimal automaton is not necessarily unique, in fact there could exist different reversible automata with the same number of states accepting the same language.

We continue the investigation of minimality in reversible automata and we will refer to the following notions. Let $\mathcal{C}$ be the family of reversible automata accepting a given language L and $A \in \mathcal{C}$:

- The automaton A is *reduced* in $\mathcal{C}$ if every automaton obtained from A by merging some equivalent states does not belong to $\mathcal{C}$.
- The automaton A is *minimal* in $\mathcal{C}$ if each automaton in $\mathcal{C}$ has at least as many states as A.
- The automaton A is the *minimum* in $\mathcal{C}$ if it is the unique (up to isomorphism) minimal automaton in $\mathcal{C}$.

Our first result is a condition that characterizes languages having several different minimal reversible automata. This condition is on the structure of the transition graph of the minimum automaton accepting the language under consideration. As a special case, we show that whenever the "irreversible part" of the minimum automaton contains a loop, it is possible to construct at least two different minimal reversible automata.

[1] From now on, we will consider only one-way automata. Hence we will omit to specify "one-way" all the times.

We also observe that there exist reversible automata that are reduced, but not minimal. Investigating this phenomenon in detail, we were able to find a language for which there exist arbitrarily large, and hence infinitely many, reduced reversible automata. Furthermore, we obtained a general construction that allows to obtain arbitrarily large reversible automata for each language accepted by a minimum deterministic automaton satisfying the structural condition given in [3] and such that the "irreversible part" contains a loop. We know that this is also possible in other situations, namely that our condition is not necessary.

Now we introduce a few preliminary notations and notions. A *deterministic automaton* (DFA) is a tuple $A = (Q, \Sigma, \delta, q_I, F)$ with the usual meaning. We allow the transition function δ to be partial and througthout the paper, we assume that all states are useful, namely they are used to accept some word. This implies that a DFA does not contain any dead state. We denote by δ^R the *reverse* transition function that associates with each state $r \in Q$ and letter $a \in \Sigma$ the set of states from which r can be reached by reading a, i.e., $\delta^R(r, a) = \{q \in Q \mid \delta(q, a) = r\}$. A state r is said to be *irreversible* when there are at least two transitions on the same letter entering r, i.e., $\#\delta^R(r, a) \geq 2$, otherwise r is *reversible*. A DFA is said to be *reversible* (REV-DFA) when each state is reversible. A language is *reversible* when there exists a REV-DFA accepting it.

A DFA A can be split in two parts: the *reversible part* and the *irreversible part*. Roughly speaking, the irreversible part consists of all states that can be reached with a path which starts in an irreversible state, and of all transitions connecting those states. The reversible part consists of the remaining states and transitions, namely the states that can be reached from the initial state by visiting *only* reversible states, and their outgoing transitions.

The above mentioned algorithm [3] for converting a minimum irreversible DFA A into an equivalent minimal REV-DFA A', if possible, keeps the same reversible part of A and creates some copies of states and transitions in the irreversible part. However, different equivalent minimal REV-DFAs might exist. (See Figure 1).

2 Minimal Reversible Automata

In this section we present a characterization of the languages having several different minimal reversible automata. From now on, let us fix a reversible language L and the minimum DFA $M = (Q, \Sigma, \delta, q_I, F)$ accepting it.

Theorem 1. *The following statements are equivalent:*

1. *There exists a state $q \in Q$ in the irreversible part such that $\delta^R(q, a) \neq \emptyset$, $\delta^R(q, b) \neq \emptyset$, for two symbols $a, b \in \Sigma$, with $a \neq b$.*
2. *There exist at least two minimal nonisomorphic* REV-DFA*s accepting L.*

As a consequence of Theorem 1 we have the following characterization of reversible languages having a unique minimal (hence a minimum) REV-DFA:

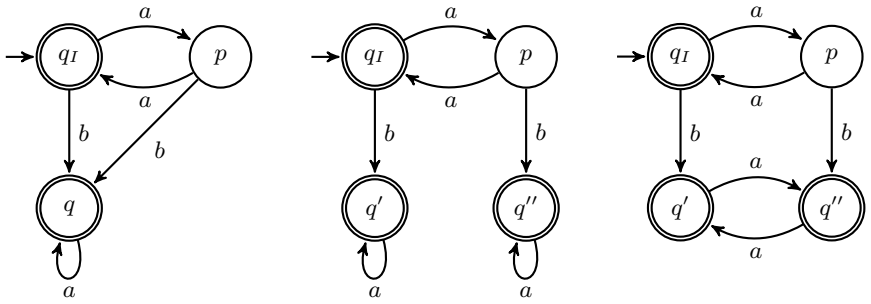


Fig. 1. A minimum DFA accepting the language $L = (aa)^* + a^*ba^*$, with two minimal nonisomorphic REV-DFAs. In the DFA on the left the reversible part consists of the states q_I and p, while the irreversible one of the state q. The REV-DFA in the center is obtained by the algorithm in [3].

Corollary 2. *There exists a unique (up to isomorphism) minimal* REV-DFA *accepting L if and only if for each state $p \in Q$ in the irreversible part, all the transitions entering in p are on the same symbol.*

We proved that when the minimum DFA accepting a reversible language contains a loop in the irreversible part the condition in Corollary 2 is always false, hence there exist at least two minimal nonisomorphic REV-DFAs. As a consequence, considering Corollary 2, we can observe that when a reversible language has a unique minimal REV-DFA, all the loops in the minimum DFA accepting it should be in the reversible part. However, the converse does not hold, namely there are languages whose minimum DFA does not contain any loop in the irreversible part, which does not have a unique minimal REV-DFA. Indeed, in [3] an example with a finite language is presented.

3 Reduced Reversible Automata

In this section, we consider reduced REV-DFAs. There exist REV-DFAs which are reduced but not minimal. Furthermore, there exist reversible languages having arbitrarly large reduced REV-DFAs and, hence, infinitely many reduced REV-DFAs.

In Figure 2 a reduced REV-DFA equivalent to the DFAs in Figure 1 is depicted. If we try to merge two states in the loop, then the loop collapses to a single state, so producing the minimum DFA, which is irreversible. Actually, this example can be modified by using a loop of N states: if (and only if) N is prime, we get a reduced automaton. This is a special case of the construction we obtained to prove the following:

Theorem 3. *If M contains a state q in the irreversible part such that the language accepted by computations starting from q is infinite, then there exist infinitely many nonisomorphic reduced* REV-DFA*s accepting L.*

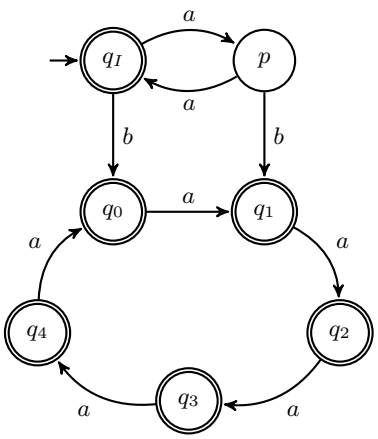


Fig. 2. A reduced REV-DFA equivalent to DFAs in Figure 1.

The condition in Theorem 3 is not necessary. In fact, we found an example where the minimum DFA does not contain any loop in the irreversible part, but it is possible to construct infinitely many equivalent reduced REV-DFAs.

References

1. Angluin, D.: Inference of reversible languages. J. ACM 29(3), 741–765 (1982)
2. Bennett, C.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
3. Holzer, M., Jakobi, S., Kutrib, M.: Minimal reversible deterministic finite automata. In: Potapov, I. (ed.) Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9168, pp. 276–287. Springer (2015)
4. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: FOCS. pp. 66–75. IEEE Computer Society (1997)
5. Kutrib, M.: Aspects of reversibility for classical automata. In: Calude, C.S., Freivalds, R., Kazuo, I. (eds.) Computing with New Resources - Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday. Lecture Notes in Computer Science, vol. 8808, pp. 83–98. Springer (2014)
6. Kutrib, M.: Reversible and irreversible computations of deterministic finite-state devices. In: Italiano, G.F., Pighizzini, G., Sannella, D. (eds.) Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9234, pp. 38–52. Springer (2015)
7. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
8. Lange, K., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. Syst. Sci. 60(2), 354–367 (2000)
9. Lombardy, S.: On the construction of reversible automata for reversible languages. In: Widmayer, P., Ruiz, F.T., Bueno, R.M., Hennessy, M., Eidenbenz, S., Conejo,

R. (eds.) Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2380, pp. 170–182. Springer (2002)
10. Pin, J.: On reversible automata. In: Simon, I. (ed.) LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings. Lecture Notes in Computer Science, vol. 583, pp. 401–416. Springer (1992)

Relating paths in transition systems: the fall of the modal mu-calculus

Catalin Dima[1], Bastien Maubert[2], and Sophie Pinchinat[3]

[1] Université Paris Est, LACL, UPEC, Créteil, France.
dima@u-pec.fr
[2] Università degli Studi di Napoli Federico II
bastien.maubert@gmail.com
[3] Université de Rennes 1, IRISA, Rennes, France.
sophie.pinchinat@irisa.fr

This is an extended abstract of a paper presented at MFCS 2015 [11].

Monadic second-order logic (MSO) is considered as a standard for comparing expressiveness of logics of programs. Ground-breaking results concerning expressiveness and decidability of MSO on infinite graphs were obtained first on "freely-generated" structures (words, trees, tree-like structures, etc.) [28,30], then on "non-free" structures like grids [18] or infinite graphs generated by regularity-preserving transformations [10,8]. In all the above settings, the syntax of MSO utilizes one or more binary relation symbols which are interpreted using the binary edge relations of the graph structure. Additionally, much attention has been brought to the study of enrichments of MSO with unary predicate symbols or with the "equal level" binary predicate (MSO^{eql}) [12,27].

For many of these settings, MSO has been compared with automata and with modal logics. Standard results on trees are Rabin's expressiveness equivalence between MSO with two successors and automata on binary trees [23], and Janin and Walukiewicz's result [16] showing that the bisimulation-invariant fragment of MSO interpreted over transition coincides with the μ-calculus. Notable exceptions to the classical trilogy between MSO, modal logics and automata are MSO on infinite partial orders – where only partial results are known [5,9,25] – and MSO^{eql}– where, similarly, only partial results are known [27].

More recently, there has been an increased interest in the expressiveness and decidability of logics defined on structures in which two "orthogonal" relations are considered: the so-called temporal epistemic (multi-agent) logics [13], which combine time-passage relations and epistemic relations on the histories of the system. Time-passage relations classically represent the evolution of the system, while each epistemic relation captures some agent's partial observation of the system by relating indistinguishable histories. They allow to reason about what these agents know about the state of the system along its executions. We may incidentally identify now an important sub-domain in verification which

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 240–244 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

is concerned with the expressivity, decidability and axiomatizability of logics of knowledge and time [15,6,29,13,14,17].

The natural question that arises regarding logics of agents that combine time and knowledge is whether a similar trilogy can be established or not. In particular, does there exist a natural extension of MSO, of the μ-calculus, and of tree automata for the temporal epistemic framework, and how would they compare? To the best of our knowledge, these questions remain open. Only partial results exist on relations between some extensions of MSO, μ-calculus, tree automata and other logics of knowledge and time [27,26,29,20].

The first observation is that appropriate extensions of MSO, of the μ-calculus and of tree automata would rely on two sorts of binary relations: those related to the behaviour of the system and those related to epistemic features. While the temporal part of these logics naturally refer to a tree-like structure, the epistemic part requires, in order to model *e.g.* powerful agents that remember the whole past, to consider binary relations defined on histories. The models of such an extension of MSO neither are tree-like structures, nor grid-like structures, nor graphs within the Caucal hierarchy. The proposals in this direction that we know about are [29,26,20] and [1]. [29] mentions an encoding of LTL with knowledge into Chain Logic with equal-level predicate, which is a fragment of MSO^{eql}. [26] introduces the epistemic μ-calculus and studies its model-checking problem. [1] studies an extension of the epistemic μ-calculus, and [20] proposes a generalization of tree automata, called jumping tree automata, which is suited to the study of temporal epistemic logics.

In this work, we develop a general setting in which models are transition systems, *i.e.* directed graphs with atomic propositions, or predicates, on vertices/states and labels on edges/transitions, together with a binary relation over their finite executions, also called paths or histories. Such relations are called *path relations*, and their definition is general enough to capture all indistinguishability relations considered in temporal epistemic logics, and more. We propose extensions of MSO and of the μ-calculus, respectively called the *monadic second order logic with path relation* and the *jumping μ-calculus*.

MSO with path relation is an extension of MSO interpreted over unfoldings of transition systems equipped with a path relation, so that a first-order variable x refers to a node in the tree-unfolding of a transition system, *i.e.* a finite execution (or path, or history). The syntax is that of MSO on graphs with an additional special binary relation symbol $\rightsquigarrow$. A formula of the form $x \rightsquigarrow y$ holds in a transition system if the path represented by x is related to the one represented by y, according to the binary relation over paths that equips the system.

The jumping μ-calculus is a generalization of the epistemic μ-calculus defined in [26]: it is also evaluated on tree-unfoldings of transition systems, and it features a *jumping modality* $\boxdot$ whose semantics relies on the path relation that equips the system. In case the path relation is seen as modelling histories' indistinguishability for some agent, this modality coincides with the classic knowledge operator K for this agent.

As in the classic setting of [16], definability in the jumping μ-calculus entails definability in MSO with path relation. It is the converse statement that we explore, that is the *expressive completeness* of jumping μ-calculus *w.r.t.* (the bisimilar invariant fragment of) MSO with path relation.

We first show that, just like alternating tree automata are equivalent to the μ-calculus, the *jumping tree automata* recently defined in [20] are equivalent to the jumping μ-calculus, and the two-way translation does not depend on the *a priori* fixed path relation. We then address, like in [16], the question whether for bisimulation-closed classes of models, definability in MSO with path relation implies definability in the jumping μ-calculus. A crucial parameter in this question is the complexity of the path relation one considers. We recall that, given a finite alphabet Σ, a binary relation over Σ^* is *regular* if there is a finite state automaton with two tapes on which it progresses synchronously (a *synchronous transducer*) that accepts a pair of words over Σ if, and only if, it is in the relation (see [2] for details). An example is the epistemic relation of an agent with synchronous perfect recall [3,4]. A relation over Σ^* is *recognizable* if there is a finite-state word automaton over $\Sigma \cup \{\#\}$, where $\#$ is a special separator symbol, that accepts precisely words of the form $w\#w'$ where (w, w') is in the relation (again refer to [2] for details). For example, epistemic relations of agents whose memory can be represented by finite state machines are recognizable relations (see [19]).

We establish the following results:

Theorem 1. *For any recognizable path relation, the jumping μ-calculus is expressive complete with respect to MSO with path relation.*

Theorem 2. *There are regular binary relations for which the jumping μ-calculus is not expressive complete with respect to MSO with path relation.*

Theorem 1 follows simply from the fact that, since recognizable relations are MSO definable, both our extensions of MSO and the μ-calculus collapse to the classic MSO and μ-calculus, respectively, when the path relation is recognizable.

Concerning transition systems with bounded branching degree, we obtain in addition that the jumping μ-calculus with recognizable path relation is at most exponentially more succinct than the μ-calculus, while its satisfiability problem is also EXPTIME-complete. These results rely on the effective translation of jumping tree automata equipped with recognizable path relations into alternating two-way tree automata [20].

To establish Theorem 2 we consider the case of the so-called *synchronous perfect recall* relation over paths [24,22], which is regular. We prove that the class of reachability games with imperfect information and perfect recall (with a fixed number of observations and actions) where the first player wins cannot be defined in the jumping μ-calculus, while being closed by bisimulation and definable in our extension of MSO. The proof heavily relies on the equivalence between the jumping μ-calculus and jumping automata, on which we exploit the "pigeon-hole principle", as well as on the use of unobservable winning conditions. Indeed, we prove that if winning conditions are assumed to be observable, then the class of

imperfect-information (either reachability or parity) games where the first player wins is definable in the jumping μ-calculus.

Our expressivity incompleteness result has several impacts.

First, we obtain that the class of jumping tree automata is not closed under projection. Indeed, the (bisimulation-closed) second-order-quantification-free fragment of MSO with path relation can be embedded into jumping tree automata. Their closure under projection would therefore imply that they coincide with the full (bisimilar invariant) MSO with path relation, which contradicts our expressivity incompleteness result.

Regarding logics of programs, there has been some interest in comparing alternating-time temporal logics with fix-point logics. When agents have perfect information, the μ-calculus subsumes these logics (see for instance [21]). For imperfect information, our results show that the picture changes: because alternating-time temporal logics with imperfect information can express the existence of winning strategies in reachability games with imperfect information, Theorem 2 reveals that the powerful jumping μ-calculus does not subsume alternating-time temporal logics with imperfect information when we consider players with perfect recall.

We also believe that our incompleteness result impacts the axiomatizability of alternating-time temporal logics with imperfect information: the impossibility to express the existence of a winning strategy in reachability games with imperfect information and perfect recall in the jumping μ-calculus strongly suggests the absence of fix-point axioms for certain alternating temporal logics.

References

1. R. Alur, P. Černý, and S. Chaudhuri. Model checking on trees with path equivalences. In *TACAS'07*, pages 664–678. Springer, 2007.
2. J. Berstel. *Transductions and context-free languages*, volume 4. Teubner Stuttgart, 1979.
3. D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and Thomas A. Henzinger. Strategy construction for parity games with imperfect information. *Inf. Comput.*, 208(10):1206–1220, 2010.
4. D. Berwanger and L. Kaiser. Information tracking in games on graphs. *Journal of Logic, Language and Information*, 19(4):395–412, 2010.
5. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theor. Comput. Sci.*, 358(2-3):150–172, 2006.
6. N. Bulling, J. Dix, and W. Jamroga. Model checking logics of strategic ability: Complexity. In M. Dastani, K. V. Hindriks, and J.-J. C. Meyer, editors, *Specification and Verification of Multi-Agent Systems*, pages 125–160. Springer, 2010.
7. N. Bulling and W. Jamroga. Alternating epistemic mu-calculus. In *Proceedings of IJCAI'2011*, pages 109–114. IJCAI/AAAI, 2011.
8. D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2003.
9. B. Courcelle and I. Durand. Automata for the verification of monadic second-order graph properties. *J. Applied Logic*, 10(4):368–409, 2012.

10. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Cambridge University Press, 2012.
11. Catalin Dima, Bastien Maubert, and Sophie Pinchinat. Relating paths in transition systems: The fall of the modal mu-calculus. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, pages 179–191, 2015.
12. C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *J. Symb. Log.*, 31(2):169–181, 1966.
13. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about knowledge*. The MIT Press, 2004.
14. J. Y. Halpern, R. van der Meyden, and M. Y. Vardi. Complete Axiomatizations for Reasoning about Knowledge and Time. *SIAM J. Comput.*, 33(3):674–703, 2004.
15. J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. 1. Lower bounds. *J. Comp. Sys. Sci.*, 38(1):195–237, 1989.
16. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proceedings of CONCUR'96*, pages 263–277. Springer, 1996.
17. A. Lomuscio and Fr. Raimondi. Mcmas: A model checker for multi-agent systems. In *Proceedings of TACAS'2006*, volume 3920 of *LNCS*, pages 450–454, 2006.
18. O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Inf. Comput.*, 179(2):356–383, 2002.
19. B. Maubert. *Logical foundations of games with imperfect information: uniform strategies*. PhD thesis, Université de Rennes 1, 2014.
20. B. Maubert and S. Pinchinat. Jumping automata for uniform strategies. In *Proceedings of FSTTCS'13*, pages 287–298, 2013.
21. S. Pinchinat. A generic constructive solution for concurrent games with expressive constraints on strategies. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *5th International Symposium on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2007.
22. B. Puchala. Asynchronous omega-regular games with partial information. In *Proceedings of MFCS'2010*, pages 592–603, 2010.
23. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, pages 1–35, 1969.
24. J.-F. Raskin, K. Chatterjee, L. Doyen, and Th. A. Henzinger. Algorithms for omega-regular games with imperfect information. *LMCS*, 3(3), 2007.
25. F. Reiter. Distributed graph automata. *CoRR*, abs/1404.6503, 2014.
26. N. V. Shilov and N. O. Garanina. Combining knowledge and fixpoints. Technical Report Preprint n.98, `http://www.iis.nsk.su/files/preprints/098.pdf`, A.P. Ershov Institute of Informatics Systems, Novosibirsk, 2002.
27. W. Thomas. Infinite trees and automaton-definable relations over omega-words. *Theor. Comput. Sci.*, 103(1):143–159, 1992.
28. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer Verlag, 1997.
29. R. van der Meyden and N. Shilov. Model checking knowledge and time in systems with perfect recall (extended abstract). In *Proceedings of FSTTCS'99*, volume 1738 of *LNCS*, pages 432–445, 1999.
30. I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.

An unifying framework for compacting Petri nets behaviors*

Giovanni Casu and G. Michele Pinna

Dipartimento di Matematica e Informatica, Università di Cagliari, Cagliari, Italy
giovanni.casu@unica.it, gmpinna@unica.it

Abstract. Compacting Petri nets behaviors means to develop a more *succinct* representation of all the possible executions of a net, still giving the capability to *reason* on properties fulfilled by the computations of the net. To do so suitable equivalences on alternative executions have to be engineered. We introduce a general notion of *merging relation*, covering the existing approaches to compact behaviors of nets, and we state some properties this kind of relations may satisfy. The classical merging relations, defined on unfoldings, do not in general satisfy the properties one may be interested in, and we propose how to add information to the executions in order to enforce some of these properties.

The behavior of a Petri net can be described in many ways, *e.g* using the *marking graph*, or the set of *firing sequences*, or its *unfolding* (see [1,2] among many others). The notion of unfolding of a net N, a net where places (called conditions) and transitions (called events) are labeled with the places and transitions of N ([3,4]), is particularly relevant as it allows to record conflicts and dependencies among the activities modeled with the Petri net N. Furthermore, the possibility of finding a finite representation of it (the prefix), has given profitability to the notion, otherwise confined to the purely theoretical modeling realm ([5,6]). However the size of a finite unfolding, even of the prefix, can be too large, hence manageable only with big efforts. Prefixes are obtained *cutting* the unfolding in such a way that each execution represented in the unfolding can be recovered in the prefix itself. The cutting procedure allows to eliminate unnecessary duplications. Still some information may be redundant, for instance the existence of conflicting components leading to isomorphic futures, but with alternative pasts, forces to represent all the possible futures, introducing in this way some avoidable duplications.

The identification of conflicting conditions seems to be a good basis for compacting nets' behaviors. Following this idea some approaches have been

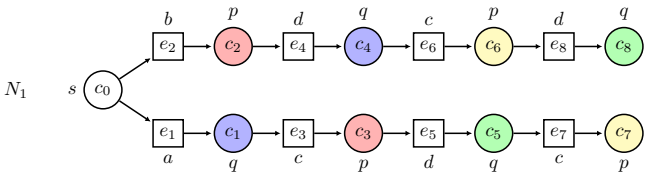
* Work partially supported by Aut. Region of Sardinia under grant P.I.A. 2013 "NOMAD" and by MIUR PRIN 2010-11 "Security Horizons"

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 245–250 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

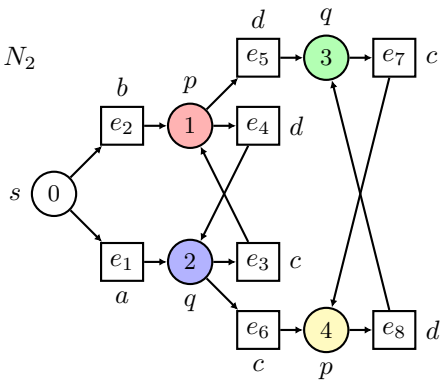
proposed, and these are based on giving precise criteria to identify conflicting conditions in nets which are *acyclic*, *i.e.* the transitive and reflexive closure of the flow relation is a partial order. In the case of *merged process* ([7]) the criterion is that the conditions must be equally labeled and have the same token occurrence (*i.e.* they represent the same token, in the collective token philosophy of [8]) whereas in the case of *trellis processes* ([9]) the criterion is the distance of the equally labeled conditions from the initial conditions (measuring the *time*). Once conditions have been identified, isomorphic *futures* can be identified as well. The identification of conflicting conditions has a *semantics* counterpart: the identification induces an equivalence relation on the different computations leading to these conditions, equivalence driven by the common futures of these computations.

We pursue this idea further, casting it in a general framework. We start choosing a representation of nets behaviors less constrained with respect to the usual notion of *causal net* on which unfoldings are based. Causal nets are acyclic safe nets where conditions may have at most one incoming arc. The uniqueness of incoming arcs, together with the safeness, guarantee that dependencies can be uniquely identified. Conflicts are deduced from conditions having more than one outgoing arcs (implying that various alternatives use that condition). We drop the assumption that each condition has at most one incoming arc, and we add the requirements that each transition in the net can be executed at most once (which is syntactically enforceable) and that restricting the net to all the transitions in a execution we obtain an acyclic net, where each condition has at most one incoming and one outgoing arc. Dependencies can be captured by looking at executions, and some conflicts may still be retrieved by looking at multiple outgoing arcs. We call these nets *unravel* nets. This notion covers the one of causal nets, as these are indeed unravel nets, whereas unravel nets may not be causal ones. Together with the notion of unravel net, we introduce a notion of conflict that it is not based on the syntax, like in causal nets, but on the semantics (the executions of the net), simply stipulating that two conditions are in conflict if they never appear together in an execution.

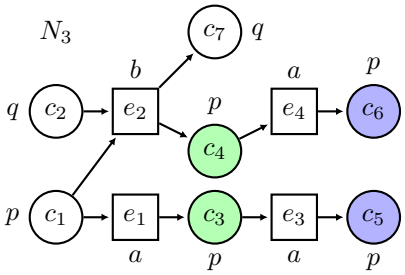
We can now put forward the general framework, that consists in taking a representation of the behaviors of a given net (in our case a labeled unravel net) and an equivalence relation defined on conditions of the chosen representation of the behaviors. The minimal requirement we put on this relation, which is called *merging relation* is that two different conditions in the relation should be *equally labeled* and in *conflict*.



Consider the unravel net N_1. Conditions c_2 and c_3 are in conflict and they have the same label p, and similarly for conditions c_1 and c_4 (here the label is q). In the net above the merging relation (denoted with $\sim$) stipulates that $c_2 \sim c_3$, $c_1 \sim c_4$, $c_6 \sim c_7$ and $c_5 \sim c_8$ (reflexive pairs omitted). The relation is identified pictorially with different colors. This is not the unique merging relation definable on this net, we could have chosen this other relation: $c_2 \sim c_7$, $c_1 \sim c_4$, $c_6 \sim c_3$ and $c_5 \sim c_8$ (again reflexive pairs omitted), and clearly the identity relation is a merging relation. Once that a merging relation is fixed, we can *compact* the behavior by merging the conditions in the same equivalence classes and identifying the equally labeled transitions having the same preset and postset. The result of this procedure is the net shown on the left. The condition 0 is the equivalence class of c_0, 1 is the equivalence class of c_2 and c_3, 2 the one of c_1 and c_4, 3 of c_6 and c_7 and finally 4 the one of c_5 and c_8. Transitions with the same labels are not identified as none of them has the same preset and postset. We observe that the net obtained identifying equivalent conditions is not any longer an unravel net. In the execution e_1 followed by e_3 and e_4 the condition 1 is marked twice violating the requirement of being acyclic. The fact that e_3 should be followed by e_5 and not e_4 has been lost in the compaction process.

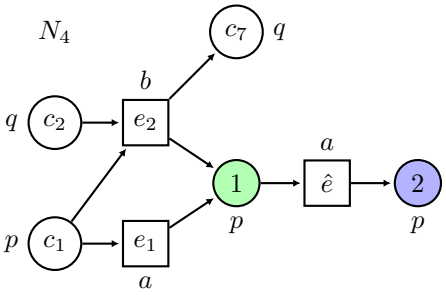


The notion of merging relation covers the criteria used in merged and trellises processes. In the case of merged and trellises processes the starting point is always a branching process, hence a labeled causal net where dependencies and conflicts can be found syntactically. The criterion to use in case of merged processes is to consider two equally labeled conflicting conditions c_i and c_j as equivalent is that they have the same *token occurrence*, which is defined as the number of conditions labeled as c_i and c_j that are encountered going back to the initial conditions, comprising c_i and c_j. In the net N_1 the conditions c_6 and the condition c_7 have both one condition in their past which has the same label, namely c_2 and c_3 respectively, hence their token occurrence is 2. In the case of trellises processes the starting point is not only a branching processes, but here the nets considered are called *multi-clocks* nets. Multi-clocks nets are the product of various automata where only one place is initially marked and each reachable marking is such that each component has



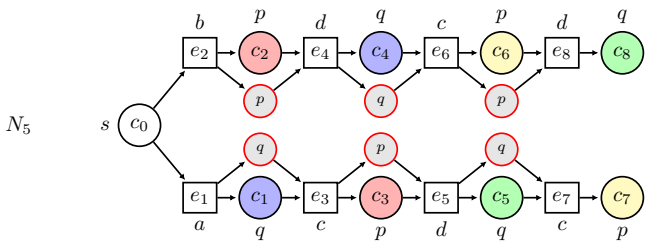
just one place marked. Due to this feature it is possible to identify, for each condition and each execution, the exact time in which the condition holds. The criterion is then the one of considering two equally labeled conflicting conditions c_i and c_j as equivalent is that they have the same *time*, which is defined as the number of conditions that are encountered going back to the initial conditions, comprising c_i and c_j. In the unravel net N_3, the conditions c_3 and c_4 have the same label p and have the same distance from the initial conditions, and similarly c_5 and c_6. By identifying these conditions also the transitions e_5 and e_6 have to be identified, resulting in the net N_4. 1 is the equivalence class containing c_3 and c_4, 2 the one with c_5 and c_6 and finally $\hat{e}$ is the transition obtained fusing e_3 and e_4, has these two transitions have the same preset, the same postset and are equally labelled, thus they *share* the same future.

The criteria used to obtain merged and trellises processes may be generalized equipping the unravel net with a mapping that associate to each condition a unique number, which we can call the *measure*. Thus a merging relation is obtained making equivalent all the conditions having the same labels, the same measure and being pairwise conflicting.

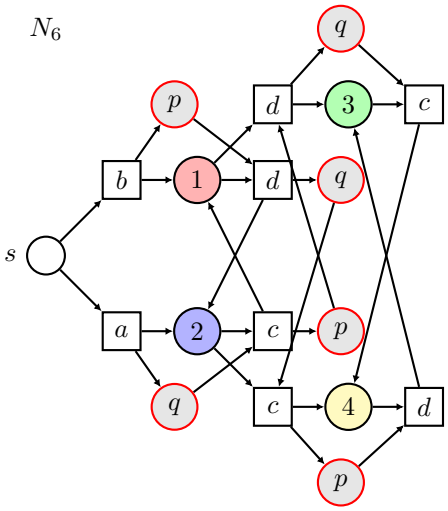


For the compaction process to be of real interest one would like to obtain a net which is possibly an unravel one, or that has strong relations with the unravel net we started with. In fact we devise two characteristic the compaction process may have. The first one is that to each execution in the compact net, at least an execution in the original one should correspond. The merging relation used to obtain N_2 out of N_1 does not fulfil this property, whereas it does the merging relation used to obtain N_4 out of N_3. When a merging relation fulfil this property we say that it is a *reflecting merging relation*. Clearly a reflecting merging relation always exists, as the identity relation is reflecting.

The property of being reflecting, adopting as the measure the token occurrence, can be enforced by enriching the starting unravel net. In the net N_1 conditions are used both to represent dependencies and conflicts, and by fusing some of them the dependencies may be lost. Thus the idea is to add some conditions that captures the dependencies. These conditions are easily obtainable by considering the whole token count for each transition of net. The net N_1 can be enriched as shown in the net N_5, and the added conditions are labeled with the condition representing the dependency.



Among the added conditions, in this case, there is no equivalence, as all of them have a different measure, the measure in this case being the one represented by the whole token count for the transitions (details on how to determine this measure can be found in [10], where the theory is applied to multi-clock nets). The result of the compaction process is the net N_6. Now the execution e_1 followed by e_3 and e_4 is no longer possible and e_3 is followed by e_5 only.



Beside looking for reflecting merging relation, one could be interested in preserving some characteristic of the net. For instance, one may be interested in preserving the fact that the resulting net is still an unravel one (and the measure induced by the time in the compaction done with trellis processes has this characteristic) or being acyclic when restricted to a certain subset of conditions (again, when considering the conditions belonging to an automata this is the case in trellis processes). When properties fulfilled by the net we start with are preserved by the compaction process we say that the merging relation is *preserving*. The merging relation giving the net N_6 preserves the property that, when only the added conditions are considered, the whole net is acyclic, and verification can be performed easily.

References

1. Desel, J., Reisig, W.: The concepts of Petri nets. Software and System Modeling **14**(2) (2015) 669–683
2. Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer (2013)
3. Winskel, G.: Event Structures. In: Petri Nets: Central Models and Their Properties. LNCS 255 (1987) 325–392
4. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica **28**(6) (1991) 575–591
5. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: CAV '92. LNCS 663 (1993) 164–177

6. Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's Unfolding Algorithm. Formal Methods in System Design **20**(3) (2002) 285–310
7. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged Processes: a new condensed representation of Petri net behaviour. Acta Informatica **43**(5) (2006) 307–330
8. van Glabbeek, R.J.: The individual and collective token interpretation of Petri nets. In: CONCUR 2005. LNCS 3653 (2005) 323–337
9. Fabre, E.: Trellis processes : A compact representation for runs of concurrent systems. Discrete Event Dynamic Systems **17**(3) (2007) 267–306
10. Casu, G., Pinna, G.M.: Flow unfolding of multi-clock nets. In: PETRI NETS 2014. LNCS 8489 (2014) 170–189

Additional Winning Strategies in Two-Player Games*

Vadim Malvone and Aniello Murano

Università degli Studi di Napoli Federico II

Abstract. We study the problem of checking whether a two-player reachability game admits more than a winning strategy. We investigate this in case of perfect and imperfect information, and, by means of an automata approach we provide a linear-time procedure and an exponential-time procedure, respectively. In both cases, the results are tight.

1 Introduction

Game theory is a powerful mathematical framework largely exploited in computer science and AI [1,9,16]. In the basic setting, we refer to two-player turn-based games where the players, conventionally named $Player_0$ and $Player_1$, play for a finite number of rounds. The states of the arena are partitioned among the players and each player can move only from the states he owns. Solving such a game amounts to check whether $Player_0$ has a *winning strategy*, i.e., a sequence of moves that allows him to achieve his goal, no matter how his opponent plays.

In several game settings having a more precise (quantitative) information about *how many* winning strategies a player has at his disposal turns out to be crucial. For example, in Nash Equilibrium, such an information amounts to solve the challenging question of checking whether the equilibrium is unique [2].

In this paper, we address the quantitative question of checking whether $Player_0$ has more than a strategy to win a finite two-player turn-based game, under the reachability condition. We consider both the cases that the players have perfect or imperfect information about the moves performed by the opponent. For the solution we use an automata-theoretic approach. Precisely, we build an automaton that accepts all tree witnesses for more than one winning strategy for $Player_0$. Hence, we reduce the addressed quantitative question to the emptiness of this automaton. In the perfect information setting, this results in a linear-time upper bound complexity. In the imperfect information setting, instead, it results in an exponential-time solution. In both cases, the results are tight.

* This work is a communication based on the works [12] and [11].

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 251–256 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

Related works. Counting strategies has been deeply exploited in *reactive* systems formal verification by means of specification logics extended with *graded modalities*, interpreted over games of infinite duration [2,6,10]. These works suitably extend preliminary reasonings in closed system verification [3,4,7].

2 The Game Model

In this section, we present our model and its semantics.

Definition 1. *A* turn-based two-player reachability game with imperfect information *(2TRGI), $Player_0$ vs $Player_1$, is a tuple* $G = <$ St, s_I, Ac, tr, W, $\cong>$, *where* $\mathrm{St} = \mathrm{St}_0 \cup \mathrm{St}_1$ *is a finite non-empty set of* states, *with* St_i *set of states of* $Player_i$, $s_I \in \mathrm{St}$ *is an* initial state, $\mathrm{Ac} = \mathrm{Ac}_0 \cup \mathrm{Ac}_1$ *is the set of actions,* W *is a set of target states,* $\mathrm{tr} : \mathrm{St}_i \times \mathrm{Ac}_i \to \mathrm{St}_{1-i}$, *for* $i \in \{0, 1\}$ *is a* transition function *mapping a state of a player and its action to a state belonging to the other player, and* $\cong = \cong_0 \cup \cong_1$ *is a set of equivalence relations on* Ac.

Let $i \in \{0, 1\}$ and $a, a' \in \mathrm{Ac}_i$ be two actions. If $a \cong_{1-i} a'$ we say that a and a' are *indistinguishable* to $Player_{1-i}$. By $[\mathrm{Ac}_i] \subseteq \mathrm{Ac}_i$ we denote the subset of actions that are *distinguishable/visible* for $Player_{1-i}$. In particular, for each set of equivalent actions over Ac_i, we put a representative action in $[\mathrm{Ac}_i]$. From any $s \in \mathrm{St}_i$, if $a \cong_{1-i} a'$ then also the reached states are indistinguishable, ie $\mathrm{tr}(s, a) = s'$ and $\mathrm{tr}(s, a') = s''$ are indistinguishable for $Player_{1-i}$. A relation $\cong_i$ is an *identity* if $a \cong_i a'$ iff $a = a'$. A *2TRGI* has perfect information (a *2TRG*, for short) if $\cong$ contains only identities. To give the semantics of *2TRGI*s, we introduce the concepts of track, strategy and play.

A *track* is a finite sequence of states $\rho \in \mathrm{St}^*$ such that, for all $i < |\rho|$, if $(\rho)_i \in \mathrm{St}_0$ then there exists an action $a_0 \in \mathrm{Ac}_0$ such that $(\rho)_{i+1} = \mathrm{tr}((\rho)_i, a_0)$, else there exists an action $a_1 \in \mathrm{Ac}_1$ such that $(\rho)_{i+1} = \mathrm{tr}((\rho)_i, a_1)$, where $(\rho)_i$ denotes the i-st element of ρ. By $last(\rho)$ we denote the last element of ρ and by $\rho_{\leq i}$ the prefix track $(\rho)_0 \dots (\rho)_i$. By $\mathrm{Trk} \subseteq \mathrm{St}^*$ we denote the set of tracks over St and by Trk_i the set of tracks ρ in which $last(\rho) \in \mathrm{St}_i$. For simplicity, we assume that all tracks in Trk start at the initial state $s_I \in \mathrm{St}$.

A *strategy* for $Player_i$ is a function $\sigma_i : \mathrm{Trk}_i \to \mathrm{Ac}_i$ that maps a track to an action. A strategy is *uniform* if it adheres on the visibility (visible actions) of the players. In the rest of the paper we only refer to uniform strategies.

The composition of strategies, one for each player, induces a computation called *play*. Precisely, assume $Player_0$ and $Player_1$ take strategies σ_0 and σ_1, respectively. Their composition *induces* a play ρ such that $(\rho)_0 = s_I$ and for each $i \geq 0$ if $(\rho)_i \in \mathrm{St}_0$ then $(\rho)_{i+1} = \mathrm{tr}((\rho)_i, \sigma_0(\rho_{\leq i}))$, else $(\rho)_{i+1} = \mathrm{tr}((\rho)_i, \sigma_1(\rho_{\leq i}))$.

We can now give the definition of *reachability winning condition*.

Definition 2. *$Player_0$ wins a 2TRGI, under the reachability condition, if he has a strategy that for all strategies of $Player_1$ the resulting induced play will enter a state in* W. *Such a strategy is said* winning *for $Player_0$.*

Given a *2TRGI* and one of its play ρ, it is possible to check who is the winner in ρ by considering only $(\rho)_0, \ldots, (\rho)_{|\mathrm{St}|+1}$. In fact, if $(\rho)_i \in \mathrm{W}$ for some $i \in \{0, \ldots, |\mathrm{St}|+1\}$ then $Player_0$ wins the play ρ, otherwise there exists a loop in which $Player_1$ can stay infinitely, and then he wins the game.

We formalize the winning condition by means of a tree structure that we call *schema strategy tree*. To proper introduce it, we provide the concept of *decision tree*. We start with some basic notion about trees.

Let Υ be a set. An *Υ-tree* is a prefix closed subset $T \subseteq \Upsilon^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. Given a node $v = y \cdot x$, with $y \in \Upsilon^*$ and $x \in \Upsilon$, we define $prf(v)$ to be y and $last(v)$ to be x. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $< T, V >$ where T is an $\Upsilon-$tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

A *decision tree* is the unwinding of the game structure along with all possible combinations of player actions, ie a tree that collects all tracks over the game. A winning strategy can be seen as an opportune mapping, over the decision tree, of a player's "strategy schema" built over the visibility. In other words, the player first makes a decision over a set S of indistinguishable states and then this choice is used in the decision tree for each state in S. This makes the decision tree *uniform*. However, observe that we use synchronous memoryfull strategies. This means that in a decision tree, the set S of indistinguishable states resides at the same level. To make this idea more precise, we now formalize the concept of *schema strategy tree* and *uniform strategy tree*.

Definition 3. *Given a 2TRGI and a uniform strategy σ for $Player_i$, a* schema strategy tree *for $Player_i$ is a $\{\top, \bot\}$-labeled $(\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}])$-tree $< T, V >$, with $T \subset (\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}])^*$ and V as follows: (i) $V(\varepsilon) = \top$; (ii) for all $v \in T$, if $last(v) \in [\mathrm{Ac}_{1-i}]$ then $V(v) = \top$, else let $\rho = (\rho)_0 \ldots (\rho)_{|v|-1}$ be a track from s_I, with $(\rho)_k = \mathrm{tr}((\rho)_{k-1}, last(v_{\leq k}))$ for each $0 \leq k \leq |v| - 1$, if $last(v) = \sigma(\rho)$ then $V(v) = \top$, else $V(v) = \bot$.*

In a schema strategy tree the $\top$ label indicates that $Player_i$ selects the corresponding set of visible states in the decision tree and $\bot$ is used conversely. In particular, the starting node of the game is the root of the schema strategy tree and it is always enabled (condition (i)); all nodes belonging to the adversarial player are always enabled; and one of the successors of $Player_i$ nodes is enabled in accordance with the uniform strategy σ (condition (ii)). Simply, a *uniform strategy tree* is a projection of the decision tree along the schema strategy tree.

3 Additional Winning Strategies for 2TRGI

In this section we provide the main result of this work. Technically, we make use of *alternating tree automata* [14].

An alternating tree automaton (*ATA*) is a tuple $A =< \Sigma, D, Q, q_0, \delta, F >$, where Σ is the alphabet, D a finite set of directions, Q the set of states, $q_0 \in Q$ the initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ the transition function, where $\mathcal{B}^+(D \times Q)$ is the set of all positive Boolean combinations of pairs (d, q) with d direction and

q state, and $F \subseteq Q$ the set of the accepting states. An *ATA* recognizes (finite) trees by means of runs. For a Σ-labeled tree $< T, V >$, with $T = D^*$, a run is a $(D^* \times Q)$-labeled $\mathbb{N}$-tree $< T_r, r >$ such that the root is labeled with (ε, q_0) and the labels of each node and its successors satisfy the transition relation. A run is *accepting* if all its leaves are labeled with accepting states. An input tree is accepted if there exists a corresponding accepting run. By $L(A)$ we denote the set of trees accepted by A. We say that A is not empty if $L(A) \neq \emptyset$.

For our purpose, we formalize the concept of *schema additional strategy tree*.

Definition 4. *Given a 2TRGI and two uniform strategies σ and σ' for $Player_i$, a* schema additional strategy tree *for $Player_i$ is a $\{\top, \bot\}$-labeled $(\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}])$-tree $< T, V >$, with $T \subset (\mathrm{Ac}_i \cup [\mathrm{Ac}_{1-i}])^*$ and V as follows: (i) $V(\varepsilon) = \top$; (ii) for all $v \in T$, if $last(v) \in [\mathrm{Ac}_{1-i}]$ then $V(v) = \top$, else let $\rho = (\rho)_0 \dots (\rho)_{|v|-1}$ be a track from s_I, with $(\rho)_k = \mathrm{tr}((\rho)_{k-1}, last(v_{\leq k}))$ for each $0 \leq k \leq |v| - 1$, if $last(v) = \sigma(\rho)$ or $last(v) = \sigma'(\rho)$ then $V(v) = \top$, else $V(v) = \bot$.*

Now, we have all ingredients to give the following result.

Theorem 1. *Given a 2TRGI the problem of deciding whether $Player_0$ has more than a uniform winning strategy is ExpTime-Complete.*

Proof. For the lower bound, we recall that 2-player turn-based games with imperfect information is *ExpTime*H [13]. For the upper bound, we use an automata approach. Precisely, we build an *ATA* A that accepts all schema additional strategy trees for $Player_0$. It has as set of states $Q = \mathrm{St} \times \mathrm{St} \times \{\top, \bot\} \times \{0, 1\} \times \{ok, split\}$ and alphabet $\Sigma = \{\top, \bot\}$. We use in Q a duplication of states as we want to remember the state associated to the parent node while traversing the tree. The flag *split* is used to remember we have to "enter" two winning strategy paths, so moving to *ok*. The flag $f \in \{1, 0\}$ indicates whether along a path we have entered or not a target state. As initial state we set $q_0 = (s_I, s_I, \top, 0, split)$. Given a state $q = (s, s', t, f, \bar{f})$, the transition relation $\delta(q, t')$ is defined as:

$$\begin{cases} \bigwedge_{a_0 \in \mathrm{Ac}_0} (d, (s', s'', \top, f', ok)) & \text{if } s' \in \mathrm{St}_0 \wedge t' = \top \wedge t = \top \wedge \bar{f} = ok \\ \bigwedge_{a_0 \in \mathrm{Ac}_0} \bigvee_{\bar{f}' \in \{ok, split\}} (d, (s', s'', \top, f', \bar{f}')) & \text{if } s' \in \mathrm{St}_0 \wedge t' = \top \wedge t = \top \wedge \bar{f} = split \\ \bigwedge_{a_1 \in \mathrm{Ac}_1} (d, (s', s'', \top, f', \bar{f})) & \text{if } s' \in \mathrm{St}_1 \wedge t' = \top \wedge t = \top \\ false & \text{if } t' = \top \text{ and } t = \bot \\ true & \text{if } t' = \bot \end{cases}$$

if $s' \in \mathrm{St}_0$ then $s'' = \mathrm{tr}(s', a_0)$ and d is in accordance with $|\mathrm{Ac}_1|$, else $s'' = \mathrm{tr}(s', a_1)$ and d is in accordance with $|\mathrm{Ac}_0|$; if $q' \in \mathrm{W}$ then $f' = 1$, otherwise $f' = f$.

The set of accepting states is $\mathrm{F} = \{(s, s', t, f, \bar{f}) : s, s' \in \mathrm{St} \wedge t = \top \wedge f = 1 \wedge \bar{f} = ok\}$. Recall that an input tree is accepted if there exists a run whose leaves are all labeled with accepting states. In our setting this means that an input tree simulates a schema additional strategy tree for $Player_0$. So, if the automaton is not empty then $Player_0$ wins the game with more than one strategy, ie, there exists a schema additional strategy tree for him.

The desired computational complexity follows by considering that: *(i)* the size of the automaton is polynomial in the size of the game, *(ii)* to check its emptiness can be performed in exponential time [5,8].

In case we study a 2*TRG* the automaton provided in the above proof sends one copy in each direction. So, the automaton is nondeterministic. By recalling that the emptiness problem in this case is solvable in linear-time [15] and the *PTime*-hardness for alternating reachability games [9] the following result holds.

Theorem 2. *Given a 2TRG the problem of deciding whether* $Player_0$ *has more than a winning strategy is PTime-Complete.*

4 Conclusion and Future Work

In this paper we have shown an automata-theoretic approach to check whether a player has more than a winning strategy in a two-player reachability game. Our approach works with optimal asymptotic complexity both in the case the players have perfect information about the moves performed by their adversarial or not. We believe that our results can be used as core engine to count strategies in more involved game scenarios and in many solution concepts reasoning. For example, it can be used to solve the *Unique Nash Equilibrium* problem, in an extensive game form of finite duration. As future work, it would be useful to check additional winning strategies in multi-agent concurrent games.

References

1. R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.
2. B. Aminof, V. Malvone, A. Murano, and S. Rubin. Graded strategy logic: Reasoning about uniqueness of nash equilibria. In *AAMAS 2016*, pages 698–706, 2016.
3. B. Aminof, A. Murano, and S. Rubin. On ctl* with graded path modalities. In *LPAR-20*, pages 281–296, 2015.
4. P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The Complexity of Enriched muCalculi. *Logical Methods in Computer Science*, 4(3):1–27, 2008.
5. E. Emerson and C. Jutla. The Complexity of Tree Automata and Logics of Programs (Extended Abstract). In *FOCS'88*, pages 328–337. IEEE Computer Society, 1988.
6. A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *Logical Methods in Computer Science*, 4(3):1–21, 2008.
7. O. Kupferman, U. Sattler, and M. Vardi. The Complexity of the Graded muCalculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.
8. O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
9. O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.
10. V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. On the counting of strategies. In *TIME 2015*, pages 170–179, 2015.

11. V. Malvone, A. Murano, and L.Sorrentino. Additional Winning Strategies in Finite Games. Under submission.
12. V. Malvone, A. Murano, and L. Sorrentino. Games with additional winning strategies. In *CILC'15*, CEUR Workshop Proceedings, vol. 1459, pages 175–180. CEUR-WS.org, 2015.
13. J. H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.
14. G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.
15. W. Thomas. Infinite trees and automaton definable relations over omega-words. In *STACS'90*, pages 263–277, 1990.
16. M. Wooldridge. *An Introduction to Multi Agent Systems.* John Wiley & Sons, 2002.

Deadlock analysis with behavioral types for actors.

Vincenzo Mastandrea[1,2]

[1] University Nice Sophia Antipolis, CNRS, I3S, UMR 7271, France
[2] INRIA - Sophia Antipolis Méditerranée

1 Introduction

Actors are a powerful computational model for defining distributed and concurrent systems [1,2]. This model has recently gained prominence, largely thanks to the success of the programming languages Erlang [3] and Scala [9]. The actor model relies on a few key principles: (a) an actor encapsulates a number of data, by granting access only to the methods inside the actor itself; (b) method invocations are *asynchronous*, actors retain a queue for storing the invocations to their methods, which are processed sequentially by executing the corresponding instances of method bodies. The success of this model originates at the same time from its simplicity, from its properties, and from its abstraction level. Indeed, programming a concurrent system as a set of independent entities that only communicate through asynchronous messages eases the reasoning on the system.

1.1 Problem: Actors and synchronizations.

Actors do not explicitly support synchronization: requests between actors are in general remote procedure calls. The only guarantee of asynchronous messages is the causal ordering created by the communication. The retrieval of the result of an asynchronous message is usually simulated by a callback mechanism where the invoker sends its identity and the invoked actor sends a result message to the invoker. However callbacks introduce an inversion of control that makes the reasoning on the program difficult. Henceforth, providing synchronization as first-class linguistic primitive is generally preferable.

Some languages extend the actor model and provide synchronizations by allowing methods to return values. In general, this is realised by using *explicit futures*. A method of an actor returns a special kind of objects called *future*; in turn the type system is extended so that some values are tagged with a *future type*. A special operation on a future allows the programmer to check whether the method has finished and at the same time retrieves the method result. The

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 257–262 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

drawback of this approach is that programmers must be aware of futures and must know how to deal with them.

We study a different extension of the actor model that uses *implicit futures* and a *wait-by-necessity* strategy: the caller synchronizes with a method invocation only when its returned value is *strictly* necessary [4]. This strategy does not require explicit synchronization operators and ad-hoc types: the scheduler stops the flow of execution when a value to be returned by a method is needed for computing an expression. The synchronization becomes data-flow oriented: if some data is accessed and this data is not yet available, the program is automatically blocked. This way, an actor can return a result containing a future without worrying about which actor will be responsible for synchronizing with the result: the synchronization will always occur as late as possible. Replacing a future by its value is no more an operation that has to be explicitly written by the programmer, as it automatically happens at some point of the computation that can be optimized by the designer of the language runtime. We defined a simple actor calculus with wait-by-necessity synchronizations, called `gASP` [6].

While synchronization is useful, if it used improperly it can cause *deadlocks* (deadlocks cannot occur in the basic actor model). Deadlock detection is a sensible issue, in particular because it is hard to verify in languages that admit systems with unbounded (mutual) recursion and dynamic actor creation.

The following example illustrates the expressiveness of (implicit) futures and the difficulties of deadlock analysis:

```
01 Int fact(Int n, Int r){
02     Act x; Int y;
03     if (n == 0) return r;
04     else {  x = new Act(); r = r*n; n = n-1;
05             y = x.fact(n,r); return y;  }}
```

The access to `fact(n,1)` boils down to exactly `n` synchronizations. Indeed, since the value of `y` is never accessed within the method, the future is returned to the caller. When accessing the value of `fact(n,1)` a synchronization is performed on the result of the first nested invocation `fact(n-1,n)` which will need to access the result of the next invocation `fact(n-1,n*n-1)`, and so on. Technically, let the type of an asynchronous invocation be called *future type*. Then the type of `fact(n,r)` is a *recursive future type*. Because of this type, it is not possible to determine at compile time how many explicit synchronizations happen when the value of `fact(x,1)` is needed, with `x` unknown.

1.2 A technique for deadlock analysis.

To address (static-time) deadlock detection of `gASP` programs, we rely on a technique that has been already used for pi-calculus [7] and for a concurrent object-oriented calculus called (`core`) `ABS` [5,8]. Our technique consists of two modules: a **front-end type (inference) system** that automatically extracts abstract behavioral descriptions relevant to deadlock analysis from `gASP` programs, called *behavioral types*, and a **back-end analyzer of types** that computes a model of dependencies between runtime entities using a fixpoint technique.

According to this technique, a synchronization between actors α and α' is modeled by a dependency pair (α, α'), which means that the termination of

a process of α depends on the termination of a process of α'. Programs are denoted by finite models that are sets of relations on names. If a circular dependency $(\alpha_1, \alpha_2) \cdots (\alpha_{n-1}, \alpha_n)(\alpha_n, \alpha_1)$ is found in one of the relations, then the corresponding program may manifest a deadlock.

Synchronization on explicit futures boils down to checking the end of a method execution and retrieving the returned object, the retrieved object can be a future itself. On the contrary, with wait-by-necessity, if a computation requires a not-yet available value then a synchronization occurs, until a proper value is available. Retrieving this value might require to wait for the termination of several methods. Indeed, consider the factorial example, let β be the actor needing the value of `fact(n,1)`. This synchronization requires that β simultaneously synchronizes with all the actors computing the nested factorial invocations, say $\beta_1, \ldots, \beta_{\mathtt{n}-1}$. A translation from `gASP` to `ABS` would require to know statically the number n of synchronisation to perform. From the analysis point of view, this means that we have to collect all the dependencies of the form $(\beta, \beta_1), (\beta, \beta_2), \ldots, (\beta, \beta_{\mathtt{n}-1})$. In [5,8], this collection was done step-by-step by generating a dependency pair for every explicit synchronization. For synchronization on implicit futures, we need to generate a sequence of dependence pair when a value is needed, and this sequence is not bound statically.

1.3 Main contribution.

Addressing adequately implicit futures amounts to define a new type system of the above program and adapt in a non-trivial way the analyzer. The challenge we address is the ability to extend the synchronization point so that an unbounded number of events can be awaited at the same time. Our solution first extends the behavioural type with *fresh future identifiers* and to introduce specific types that identify whether a future is synchronised or not. A method signature also declares the set of actors and futures it creates to handle the potential unbounded number of future and actor creations. Then, we exploit the relation that exists between the number of dependencies of a synchronization and the number of nested method invocations. Instead of associating dependencies to synchronization points, *we delegate the production of the dependencies to method invocations*, each contributing with its own dependency. The sequence of dependencies is unfolded during the analysis. To implement this methods types of `gASP` carry an additional formal parameter, called *handle*, which is instantiated by the actor requiring the synchronization when this happens. The evaluation of behavioural types in the analyzer also carries an environment binding future names to their values (method invocations).

2 Behavioral Types

The deadlock detection technique we present uses abstract descriptions, called *behavioral types*, that are associated to programs by a type system. The purpose of the type system is to collect dependencies between actors and between futures and actors. At each point of the program, the behavioral type gathers informations on

local synchronizations and on actors potentially running in parallel. We perform such an analysis for each method body, gathering the behavioral information at each point of the program.

A *behavioral type program* is a pair $\left(\mathcal{L}, \Theta \cdot \mathtt{L}\right)$, where $\mathcal{L}$ is a *finite set* of *method behaviors* $\mathtt{m}(\alpha, \overline{\mathtt{x}}, X) = (\nu\, \overline{\varphi})(\Theta_{\mathtt{m}} \cdot \mathtt{L}_{\mathtt{m}})$, with $\alpha, \overline{\mathtt{x}}, X$ being the *formal parameters* of $\mathtt{m}$, $\Theta_{\mathtt{m}}$ the *future environment* of $\mathtt{m}$, $\mathtt{L}_{\mathtt{m}}$ the behavioral types for the *body* of $\mathtt{m}$, and Θ and $\mathtt{L}$ are the *main future environment* and the *main behavioral type*, respectively. A future environment Θ maps future names to future behaviors (without synchronization information) $\lambda X.\mathtt{m}(\alpha, \overline{\mathtt{x}}, X)$. In the method behavior, the formal parameter α corresponds to the identity of the object on which the method is called (the `this`), while X, called *handle*, is a place-holder for the actor that will synchronize with the method. In practice several actors can synchronise with the same future, but only one at a time. $\overline{\mathtt{x}}$ are the type of the method parameters. The binder $(\nu\overline{\varphi})$ binds the occurrences of $\overline{\varphi}$ in $\Theta_{\mathtt{m}}$ and $\mathtt{L}_{\mathtt{m}}$, with φ ranging over future or actor names.

The basic types $\mathtt{r}$ are used for values: they may be either $\Box$, to model integers, or any actor name α. The extended type $\mathtt{x}$ is the type of variables, and it may be a value type $\mathtt{r}$ or a *not-yet-synchronized type* $\mathtt{r}_f$ (in order to retrieve the value $\mathtt{r}$ it is necessary to synchronize the future f). The behavioral type $\mathtt{0}$ enforces no dependency, (κ, α) enforces the dependency between κ and α meaning that, if κ is instantiated by an actor β, β will need α to be available in order to proceed its execution. f_κ may represent different behaviors depending on the value of κ: $f_\star$ represents an unsynchronized future f, which is a pointer in the future environment to the corresponding method invocation; f_α represents the synchronization of the actor α with the future f; f_X represents the return of a future f by the method associated to the handler X. The type $\mathtt{L} \,\&\, \mathtt{L}'$ is the *parallel composition* of $\mathtt{L}$ and $\mathtt{L}'$, it is the behavior of two methods running in parallel and not necessarily synchronized. The sum $\mathtt{L} + \mathtt{L}'$ it is the composition of two behaviors that cannot occur at the same time, either because one occurs before the other or because they are exclusive.

In general, a statement has a behavior which is a sum of behaviors. Each term of the sum is a parallel composition of synchronization dependencies and unsynchronized behaviors. We propagate this way the set of methods running in parallel as a set of not-yet-synchronized futures all along the type analysis. The statements that create no synchronization at all (i.e. that do not access a future, nor call a method, nor return from a method) have behavior 0.

Example. The behavioral type associated to the following program is $(\mathtt{fact\_d}(\alpha, \Box_f, X) = (\nu\, f')(\Theta_{\mathtt{fd}} \cdot \mathtt{L}_{\mathtt{fd}}), \Theta \cdot \mathtt{L})$.

```
01 Int fact_d(Int n){
02    Int y;
03    if (n == 0) return 1;
04    else { n = n-1; y = this.fact_d(n);
05           y = y*(n+1) ; return y; }}
```

$$\begin{aligned} \Theta_{\mathtt{fd}} &= \{f' \mapsto \lambda X.\ \mathtt{fact\_d}(\alpha, \Box, X)\} \\ \mathtt{L}_{\mathtt{fd}} &= (f_\alpha + f'_\star + f'_\alpha)\ \&\ (X, \alpha) \\ \Theta &= \{f'' \mapsto \lambda X.\mathtt{fact\_d}(\alpha, \Box, X)\} \\ \mathtt{L} &= f''_\star + f''_{main} \end{aligned}$$

The synchronization f'_α, contained in the behavior $\mathtt{L}_{\mathtt{fd}}$, causes a deadlock. The corresponding method invocation $(\lambda X.\ \mathtt{fact\_d}(\alpha, \Box, X))$ is performed on the actor α, which amounts to instantiate the pair (X, α) into (α, α).

3 Future work.

Relaxing constraints. In order to simplify our arguments, we focussed on a sublanguage where futures are either returned or synchronized within a method body. This implies that a synchronization on a method will cause the simultaneous synchronization on every new future it may have directly or indirectly triggered. More specifically, after the synchronization we are guaranteed that every other method invocation triggered by it has terminated. We intend to relax this restriction by admitting method behaviours that trigger unsynchronized tasks. We already investigated this extension in [8] and the application to `gASP` of the solutions therein seems possible. A similar remark concerns the restriction that fields of actors must be ground integers. We can relax it by using records, as we did in [8]. In this case, the problematic issue will be to admit fields that store futures while keeping the precision of the analysis acceptable.

Actor model extension. A possible evolution of our work could be continue the study of deadlock analysis on some actor model extension. Generally an actor runs a single applicative thread, but in [10] a version of the model in which each actor is able to run more than one thread in parallel is presented. This extension both enhances efficiency on multicore machines, and prevents most of the deadlocks of the actors. It is trivial to see that if there are no constraints related to the number of methods that can run in parallel on the same actor, the model results to be deadlock free. However, it could be interesting to study this extension of the actor model enriched by a concept that can be defined as *compatibility between methods*. This compatibility can be a property defined by the programmer that through some kind of annotation can specify if it is *safe* to run in parallel some methods. In this context safe can mean that there are no data races condition if the compatible methods are executed in parallel on the same actor. We think that our technique can be applied on this scenario extending the type system in order to express the compatibility concept.

References

1. G. Agha. The structure and semantics of actor languages. In *REX Workshop*, pages 1–59, 1990.
2. G. Agha, I. Mason, S. Smith, and C. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.
3. J. Armstrong. Erlang. *Communications of ACM*, 53(9):68–75, 2010.
4. D. Caromel, L. Henrio, and B. P. Serpette. Asynchronous sequential processes. *Inf. Comput.*, 207(4):459–495, 2009.
5. E. Giachino, C. A. Grazia, C. Laneve, M. Lienhardt, and P. Y. H. Wong. Deadlock analysis of concurrent objects: Theory and practice. In *Proceedings of IFM 2013*, volume 7940 of *LNCS*, pages 394–411. Springer, 2013.
6. E. Giachino, L. Henrio, C. Laneve, and V. Mastandrea. Actors may synchronize, safely! In *PPDP, Sep 2016, Edinburgh, United Kingdom. (to appear)*, 2016.
7. E. Giachino, N. Kobayashi, and C. Laneve. Deadlock analysis of unbounded process networks. In *Proceedings of CONCUR 2014*, volume 8704, pages 63–77.

8. E. Giachino, C. Laneve, and M. Lienhardt. A framework for deadlock detection in core ABS. *Software and Systems Modeling*, 2015.
9. P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 410(2-3):202–220, 2009.
10. L. Henrio, F. Huet, and Z. István. Multi-threaded active objects. In C. Julien and R. De Nicola, editors, *COORDINATION'13*, LNCS. Springer, June 2013.

On the Clustered Shortest-Path Tree Problem
(Short Communication)

Mattia D'Emidio[1], Luca Forlizzi[2], Daniele Frigioni[2], Stefano Leucci[3], Guido Proietti[2,4]

[1] Gran Sasso Science Institute (GSSI), Viale F. Crispi 7, I–67100 L'Aquila, Italy.
mattia.demidio@gssi.infn.it
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Via Vetoio, I–67100 L'Aquila, Italy.
{luca.forlizzi,daniele.frigioni,guido.proietti}@univaq.it
[3] Dipartimento di Informatica "Sapienza" Università di Roma, Viale R. Elena 295b, I–00161 Roma, Italy. leucci@di.uniroma1.it
[4] Istituto di Analisi dei Sistemi e Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche, Via dei Taurini 19, I–00185 Roma, Italy.

Abstract. Given an n-vertex and m-edge non-negatively real-weighted graph $G = (V, E, w)$, whose vertices are partitioned into a set of k clusters, a *clustered network design problem* on G consists of finding a (possibly optimal) solution to a given network design problem on G, subject to some additional constraint on its clusters. In this paper, we focus on the classic *shortest-path tree* problem and summarize our ongoing work in this field. In particular, we analyze the hardness of a clustered version of the problem in which the additional feasibility constraint consists of forcing each cluster to form a (connected) subtree.

1 Introduction

In several network applications, the underlying set of nodes may be partitioned into *clusters*, with the intent of modeling some aggregation phenomena taking place among similar entities in the network. In particular, this is especially true in communication and social networks, where clusters may refer to local-area subnetworks and to communities of individuals, respectively. While on one hand the provision of clusters allows to represent the complexity of reality, on the other hand it may ask for introducing some additional constraints on a feasible solution to a given network design problem, with the goal of preserving a specific cluster-based property. Thus, on a theoretical side, given a vertex-partitioned input (possibly weighted) graph G, a *clustered* (a.k.a. *generalized*) *network design problem* on G consists of finding a (possibly optimal) solution to a given network design problem on G, subject to some additional constraint on its clusters.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 263–268 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

One of the most intuitive constraint one could imagine is that of maintaining some sort of *proximity* relationship among nodes in a same cluster. This scenario has immediate practical motivations: for instance, in a communication network, this can be convincingly justified with the requirement of designing a network on a classic two-layer (i.e., local *versus* global layer) topology. In particular, if the foreseen solution should consist of a (spanning) tree T in G, then a natural setting is that of forcing each cluster to induce a (connected) subtree of T. For the sake of simplicity, in the following this will be referred to as a *clustered tree design problem* (CTDP), even if this is a slight abuse in the nomenclature. As a consequence, classic tree-based problems on graphs can be revisited under this new perspective, and while some of them do not actually exhibit, from a computational point of view, a significant misbehavior w.r.t. the ordinary (i.e., non-clustered) counterpart (for instance, the *minimum spanning tree* (MST) problem falls in this category, since we can easily solve its clustered version by first computing a MST of each cluster, then contracting these MSTs each to a vertex, and finally finding a MST of the resulting graph), some other will actually become much more complex, as it is the case for the problem of our interest in this paper, namely the *single-source shortest-path tree* (SPT) problem.

Related Work. Several classic tree/path-based problems have already been investigated in the framework of CTDPs. For instance, we refer the reader: (i) to [1,4] for studies that have focused on the clustered *traveling salesperson problem*; (ii) to [2,6] for works that have dealt with the clustered version of the *minimum Steiner tree problem*. Moreover, we mention a study that has tackled the clustered variant of the *minimum routing-cost spanning tree problem* [5], where the authors also present an inapproximability result for the clustered *shortest path problem*, which was in fact inspiring our present study. Finally, we refer the reader to the paper by Feremans *et al.* [3], where the authors review several classic network design problems in a clustered perspective, but with different side constraints on the clusters.

Our Contribution. In this paper, we focus on the clustered version of the SPT (say CluSPT in the following), and on its unweighted variant (say CluBFS in the following, where BFS refers to the *breadth-first search tree*). It is worth noticing that an SPT supports a set of communication primitives of primary importance, as for instance the broadcasting and the spanning tree protocol, and that in a non-clustered setting it can be computed in almost linear time by means of the classic Dijkstra's algorithm. Nevertheless, to the best of our knowledge nothing is known about its clustered variant, despite the fact that it is very reasonable to imagine a scenario where the aforementioned primitives are required to be applied locally and hierarchically within each cluster. In this work, we then try to fill this gap, by providing a set of results which allow to shed light on its computational complexity.

Graph Notation. Throughout the paper, we use the following graph notation. Let $G = (V, E, w)$ denote a generic *weighted* undirected graph with $|V| = n$ vertices

and $|E| = m$ edges, where $w : E \to \mathbb{R}_{\geq 0}$ is a weight function, associated with the graph, such that each edge $e = (u, v) \in E$ has a non-negative weight $w(e)$. We denote by $N(u)$ the set of neighbors of u in G, i.e., $N(u) = \{v \in V \mid (u, v) \in E\}$. Let $d_G(u, v)$ denote the *distance* between vertices u and v in G, that is the *length* of a shortest path $P_G(u, v)$ between u and v in G, which is given by the sum of the weights of the edges in $P_G(u, v)$. For a given spanning tree T of G, $d_T()$ will denote the corresponding distance function on T. Given a subset of vertices $S \subseteq V$ of G, we denote by $G[S]$ the subgraph of G induced by S. Finally, we denote by $V(G)$ and $E(G)$ the set of vertices and edges of G when we need to emphasize the dependence on the graph.

2 CluBFS

In this section, we formally introduce the CluBFS problem and then give our main results about it. The problem is defined as follows.

CluBFS	
***Input*:**	An unweighted undirected graph $G = (V, E)$, whose set of vertices is partitioned into a set of k (pairwise disjoint) clusters $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$, a distinguished source vertex $s \in V$.
***Solution*:**	A clustered BFS tree of G rooted at s, i.e., a spanning subgraph T of G such that: (i) T is a spanning tree of G rooted at s; (ii) for each $V_i \in \mathcal{V}$, $T[V_i]$ is connected.
***Measure*:**	The cost of T, i.e., $\text{COST}(T) = \sum_{v \in V} d_T(s, v)$.

In other words, a clustered BFS tree is a spanning tree T of G such that each subgraph $T_i = T[V_i]$ is connected and the sum of the hop distances in T from the source s towards all the other vertices is minimized. By a reduction from the NP-complete 3–CNF–SAT problem, we are able to prove the following result:

Theorem 1. CluBFS *is NP-hard.*

2.1 An approximation algorithm for CluBFS

In this subsection, we provide an approximation algorithm for the CluBFS problem. The main idea of the algorithm is that of minimizing the number of distinct clusters that must be traversed by any path from s to a vertex $v \in V$. If all the clusters are of low diameter then this leads to a good approximation for CluBFS. If at least one cluster has large diameter then it is possible to show that the optimal solution must be expensive and hence any solution for CluBFS will provide the sought approximation.

W.l.o.g., let V_1 be the cluster containing vertex s. The algorithm first considers each cluster $V_i \in \mathcal{V}$ and identifies all the vertices belonging to V_i into a single vertex ν_i, so as to obtain a graph G' in which (i) each vertex corresponds to a cluster and (ii) there is an edge (ν_i, ν_j) between two vertices in G' iff the set $E_{i,j} = \{(v_i, v_j) \in E(G) : v_i \in V_i \wedge v_j \in V_j\}$ is not empty. It then computes a

BFS tree T' of G' rooted at ν_1 and constructs the sought approximate solution $\widetilde{T}$ as follows: initially $\widetilde{T}$ contains all the vertices of G and the edges of a BFS tree of $G[V_1]$ rooted at s; then, for each edge (ν_i, ν_j) of T' where ν_i is the parent of ν_j in T', it adds to $\widetilde{T}$ a single edge $(v_i, v_j) \in E_{i,j}$ along with all the edges of a BFS tree of $G[V_j]$ rooted at v_j.

The analysis of the above algorithm allows us to prove the following result:

Theorem 2. *There exists a polynomial-time $O(n^{\frac{2}{3}})$-approximation algorithm for* CluBFS.

While CluBFS thus admits an $o(n)$-approximation algorithm, interestingly we proved (by a reduction from the NP-complete Exact–Cover–by–3-Sets problem) that the clustered *single-source to single-destination* shortest-path problem (on unweighted graphs) cannot be approximated in polynomial time within a factor of $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$. This extends the inapproximability result (within any polynomial factor) that was given in [5] for the corresponding weighted version. Thus, establishing an $o(n^{2/3})$-inapproximability of CluBFS is a problem that we leave open.

2.2 Fixed-Parameter Tractability Results for CluBFS

In this subsection, we prove that CluBFS is fixed-parameter tractable w.r.t. two natural parameters by providing two different FPT algorithms. The notion of *fixed-parameter tractability* relaxes the classical notion of polynomial-time tractability, by admitting algorithms whose running time is exponential, but only in terms of some parameter of the problem instance that can be expected to be small in typical applications.

In the first algorithm, we choose as our first "natural" parameter the number of clusters of $\mathcal{V}$. Notice that every solution T for CluBFS induces a *cluster-tree* $\widetilde{T}$ obtained from T by identifying the vertices belonging to the same cluster. The algorithm first guesses the cluster-tree $\widetilde{T}^*$ of an optimal solution T^*, and then it reconstructs T^* by using a dynamic programming approach. Notice that our first FPT algorithm is efficient when the number of clusters of the CluBFS instance is small. On the other hand, the classical BFS tree problem can be seen as a special instance of CluBFS where $\mathcal{V} = \{\{v\} : v \in V\}$, i.e., each cluster contains only one vertex. This problem can clearly be solved in polynomial time, but the complexity of the above algorithm becomes super-exponential! This suggests that, for the case in which $\mathcal{V}$ consists of many singleton clusters, there must be another parametrization yielding a better complexity. Following this observation, we are able to develop another FPT algorithm parameterized in the total number of vertices, say h, that belong to clusters of size at least two. The idea of the algorithm is that of guessing the *cluster-root* r_i of each cluster $V_i \in \mathcal{V}$, i.e., a vertex of V_i closer to the source s in an optimal solution T^* to the CluBFS instance. It can be shown that $T^*[V_i]$ must be a BFS tree of $G[V_i]$ rooted at r_i, and this, along with the knowledge of the cluster-roots, allows us to efficiently reconstruct the optimal tree T^*. Thus, overall, we can give the following result:

Theorem 3. CluBFS *can be solved in* $O(\min\{n\,k^{k-2}, h^h\}\cdot(m+n))$ *time and* $O(m)$ *space.*

3 CluSPT

In this section, we give our results on the CluSPT problem. Regarding the formal definition of CluSPT, it can be simply derived as the weighted version of CluBFS. In more details, the main differences w.r.t. CluBFS are then two: (i) the given graph $G = (V, E, w)$ is weighted by a weight function $w : E \to \mathbb{R}_{\geq 0}$; (ii) the measure that we are willing to minimize (i.e. the cost of T) is expressed in terms of distances (instead of hop distances, as in the unweighted case). In other words, in this case, a clustered SPT is a spanning tree T of G such that each subgraph $T_i = T[V_i]$ is connected, and the total length of all paths in T emanating from the source s is minimized. By elaborating on the reduction we used to prove the NP-hardness of CluBFS, we are able to prove the following:

Theorem 4. CluSPT *cannot be approximated, in polynomial time, within a factor of* $n^{1-\epsilon}$ *for any constant* $\epsilon \in (0, 1]$*, unless* $\mathsf{P} = \mathsf{NP}$*.*

The above result is easily seen to be (essentially) tight, since we can provide a simple $O(n)$-approximation algorithm, as follows. First it computes a multigraph G' from G by identifying each cluster $V_i \in \mathcal{V}$ into a single vertex ν_i. When doing this, it associates each edge of G' with the corresponding edge of G. Then it computes a *minimum spanning tree* (MST from now on) T' of G', and k MSTs $T_1, \ldots, T_k$ of $G[V_1], \ldots, G[V_k]$, respectively. Finally, the algorithm returns the spanning tree $\widetilde{T}$ of G which contains all the edges in $E' \cup \bigcup_{i=1}^{k} E(T_i)$, where E' denotes the set of edges of G associated with an edge in $E(T')$.

3.1 Fixed-Parameter Tractability Results for CluSPT

The two fixed-parameter algorithms for CluBFS, presented in Section 2.2, can be easily extended to CluSPT. In particular, the theorem below can be easily derived by Theorem 3 basically by replacing, in both the algorithms for CluBFS, the BFS algorithm with the Dijkstra's algorithm, when the part of the solution to the problem inside each cluster has to be computed.

Theorem 5. CluSPT *can be solved in* $O(\min\{n\,k^{k-2}, h^h\}\,(m+n\log n))$ *time and* $O(m)$ *space.*

References

1. Xiaoguang Bao and Zhaohui Liu. An improved approximation algorithm for the clustered traveling salesman problem. *Inf. Process. Lett.*, 112(23):908–910, 2012.
2. Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6, 2013.

3. Corinne Feremans, Martine Labbé, and Gilbert Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.
4. Nili Guttmann-Beck, Refael Hassin, Samir Khuller, and Balaji Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.
5. Chen-Wan Lin and Bang Ye Wu. On the minimum routing cost clustered tree problem. *J. Comb. Optim.*, 31(1):1–16, 2016.
6. Bang Ye Wu and Chen-Wan Lin. On the clustered Steiner tree problem. *J. Comb. Optim.*, 30(2):370–386, 2015.

Influence Maximization in the Independent Cascade Model

Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj

Gran Sasso Science Institute (GSSI), Viale F. Crispi, 7, 67100, L'Aquila, Italy.
{gianlorenzo.dangelo,lorenzo.severini,yllka.velaj}@gssi.infn.it

Abstract. We present our ongoing work on the problem of increasing the information spread in a network by creating a limited amount of new edges incident to a given initial set of active nodes. As a preliminary result, we give a constant approximation algorithm for the case in which the set of initial active nodes is a singleton. Our aim is to extend this result to the general case. We outline some further research directions which we are investigating.

1 Introduction

Studying the processes by which ideas and influence propagate through a network has been one of the main goals in the field of social network analysis. The influence problem is motivated by many applications in different fields: from marketing, with the aim of maximizing the adoption of a new product [3], to epidemiology, in order to limit the diffusion of a virus or disease [11], going through the analysis of social networks to find influential users and to study how information flows through the network [1].

Different models of information diffusion have been introduced in the literature [5], two widely studied models are: the *Linear Threshold Model* (LTM) and the *Independent Cascade Model* (ICM). In both models, we can distinguish between active, or infected, nodes, called seeds, which spread the information, and inactive ones. Recursively, currently infected nodes can infect their neighbours with some probability. After a certain number of such cascading cycles, a large number of nodes becomes infected in the network. In LTM the idea is that a node becomes active if a large part of its neighbours is active. More formally, each node u has a threshold t chosen uniformly at random in the interval $[0, 1]$. The threshold represents the fraction of neighbours of u that must become active in order for u to become active. At the beginning of the process a small percentage of nodes of the graph is set to active in order to let the information diffusion

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 269–274 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

process start. In subsequent steps of the process a node becomes active if the fraction of its active neighbours is greater than its threshold. In ICM, instead, a seed u tries to influence one of its inactive neighbours but the success of node u in activating the node v only depends on the propagation probability of the edge from u to v (each edge has its own value). Regardless of its success, the same node will never get another chance to activate the same inactive neighbour. The process terminates when no further node gets activated.

An interesting question, in the analysis of the information spread through a network, is how to shape a given diffusion process so as to maximize or minimize the number of activated nodes at the end of the process by taking intervention actions. Many intervention actions have been studied in the literature, the most important one is: if we are allowed to add at most k seeds, which ones should be selected so as to maximize the number of active nodes resulting from the diffusion process [5]. Besides source selection, other intervention actions may be used to facilitate or limit the diffusion processes, such as inserting or deleting edges and adding or deleting nodes in the network.

To the best of our knowledge, under LTM, the problems that have been studied are the following: Khalil et al. [6] consider two types of actions, adding edges to or deleting edges from the existing network and they show that this network structure modification problem has a supermodular objective and therefore can be solved by algorithms with provable approximation guarantees. Zhang et al. [15] consider arbitrarily specified groups of nodes, and interventions that involve both edge and node removal from the groups. They develop algorithms with rigorous performance guarantees and good empirical performance. Kimura et al. [7] use a greedy approach to delete edges under the LTM without any analysis of the supermodularity of the objective, nor rigorous approximation guarantees. Kuhlman et al. [9] propose heuristic algorithms for edge removal under a simpler deterministic variant of LTM which is not only hard, but also has no approximation guarantee. Papagelis [12] and Crescenzi et al. [4] study the problem of augmenting the graph in order to increase the connectivity or the centrality of a node, respectively and experimentally show that this increases the expected number of eventual active nodes. Under ICM, the main results are the following: Wu et al. [14] consider intervention actions other then edge addition, edge deletion and source selection, such as increasing the probability that a node infects its neighbours. It can be shown that optimizing the selection of such actions with a limited budget tends to be NP-hard and is neither submodular nor supermodular. Sheldon et al. [13] study the problem of node addition to maximize the spread of information, and provide a counterexample showing that the objective function is not submodular. Bogunovic [2] addresses the node deletion problem providing a greedy algorithm. Kimura et al. [8] propose methods for efficiently finding good approximate solutions on the basis of a greedy strategy for the edge deletion problem under the ICM, but do not provide any approximation guarantees.

In this paper, we focus on the Independent Cascade Model and investigate the problem of adding a small number of edges incident to an arbitrary seed in order to increase the spreading of information in terms of number of nodes

that become active. Thus, the problem we analyse differs from above mentioned ones and, as far as we know, similar problems have never been studied for the Independent Cascade Model.

The aim of this paper is reporting our ongoing research on which we wish to get feedback so as to possibly include these results in future publications.

2 Preliminary results

In this section we will give all the necessary definitions, introduce the problem that will be considered and show our preliminary results.

A social network is represented by a weighted directed graph $G(V, \hat{a}\check{A}\check{E}E, p)$ where V represents the set of nodes, E represents set of relationships and $p : V \times V \to [0, 1]$ is the probability of an edge to propagate information. For each node u, N_u denotes the set of neighbours of u, i.e. $N_u = \{v|(u, v) \in E\}$.

The Independent Cascade Model [5] is an information diffusion model where the information flows over the network through cascade. Nodes can have two states, active: it means the node is already influenced by the information in diffusion, inactive: node is unaware of the information or not influenced. The process runs in discrete steps. At the beginning of ICM process, few nodes are given the information, they are known as seed nodes. Upon receiving the information these nodes become active. In each discrete step, an active node tries to influence one of its inactive neighbours. Regardless of its success, the same node will never get another chance to activate the same inactive neighbour. The success of node u in activating the node v depends on the propagation probability of the edge (u, v) defined as p_{uv}, each edge has its own value. The process terminates when no further node gets activated.

We define the influence of a set $A \subseteq V$ in the graph G, denoted by $\sigma(A, G)$, to be the expected number of active nodes at the end of the process, given that A is the initial set of seeds. Given a set S of edges not in E, we denote by $G(S)$ the graph augmented by adding the edges in S to G, i.e. $G(S) = (V, E \cup S)$.

Given a graph $G = (V, E)$, a vertex set $A \subseteq V$ and an integer k, the problem we are studying consists in finding a set S of edges incident to the nodes in A not in E (that is, $S \subseteq \{(a, v) : v \in V \setminus N_a, a \in A\}$) such that $|S| \leq k$ and $\sigma(A, G(S))$ is maximum.

In the paper we focus on the case $A = \{a\}$. We leave the case $|A| > 1$ as a future work. It has been shown [10] that for a monotone submodular function the following greedy algorithm provides a $\left(1 - \frac{1}{e}\right)$-approximation: start with the empty set and repeatedly add an element that gives the maximal marginal gain. The greedy algorithm can be extended to any monotone submodular objective function thanks to the following result.

Theorem 1 ([10]). *For a non-negative, monotone submodular function f, let S be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the value of f. Then S provides a $\left(1 - \frac{1}{e}\right)$-approximation.*

In this paper, we exploit this result by showing that $\sigma(A, G(S))$ is monotone and submodular w.r.t. the possible set of edges incident to a.

Theorem 2. *$\sigma(A, G(S))$ is a monotonically increasing submodular function of the set S of edges to be added.*

Proof (sketch). We will use the definition of live-edge graph $X = (V, E_X)$ which is a directed graph where the set of nodes is equal to V and the set of edges is a subset of E. E_X is given by a edge selection process such that each edge is either live or blocked according to its propagation probability. We can assume that for each pair of neighbours in the graph, a coin of bias p_{uv} is flipped and the edges for which the coin indicated an activation are live, the remaining are blocked. It is easy to show that a diffusion model is equivalent to the reachability problem in live-edge graphs: given any seed set A, the distribution of active node sets after the diffusion process ends is the same as the distribution of node sets reachable from A in a live-edge graph.

We denote with $\chi(G)$ the probability space in which each sample point specifies one possible set of outcomes for all the coin flips on the edges, it is the set of all possible live-edge graphs. Let $R(A, X)$ denote the set of all nodes that can be reached from the nodes in A on a path consisting entirely on live edges: $R(A, X) = \bigcup_{a \in A} R(a, X)$.

The main idea to prove that the function in monotonically increasing is that, after an edge addition in G, the live-graph X has at least one more edge than the original live-edge graph, hence, the number of reachable nodes can not decrease. To prove submodularity, we note that the number of new reachable nodes from the seed after the edge addition in $G(T)$ is smaller or equal than the number of new reachable nodes in $G(S)$ since most of the nodes are already reachable by the edges in $T \setminus S$. We prove these conditions for all $X \in \chi(G)$. □

Note that, in the problem we are studying, the greedy algorithm can not evaluate the influence function exactly since $\sigma(A, G(S))$ is the expected number of activated nodes and it has been proven that evaluating this function is generally $\#P$-complete for ICM [3]. However, by simulating the diffusion process sufficiently many times and sampling the resulting active sets, it is possible to obtain arbitrarily good approximations to $\sigma(A, G(S))$ (see Prop 4.1 in [5] to bound the number of samples needed to obtain a $(1 + \delta)$-approximation). It is an extension of the result of Nemhauser et al. [10] that by using $(1 \pm \delta)$-approximate values for the function to be optimized where $\delta \geq 0$, we obtain $\left(1 - \frac{1}{e} - \epsilon\right)$-approximation, where ϵ depends on δ and goes to 0 as $\delta \to 0$.

Theorem 3. *For the problem of adding a set S of edges, not in E, incident to the node in $A = \{a\}$ such that $|S| \leq k$ and $\sigma(A, G(S))$ is maximum, there is a polynomial-time algorithm approximating the maximum influence to within a factor of $\left(1 - \frac{1}{e} - \epsilon\right)$ where ϵ is any positive real number.*

3 Future research

In this paper, we presented our ongoing work on the problem of increasing the information spread in a network considering the case in which the set of active nodes A is a singleton. We have analysed the properties of the influence function which is monotonically increasing and submodular and we propose a greedy approximation algorithm for efficiently computing a set of edges that a seed can decide to add to the graph in order to increase the expected number of influenced nodes. As future works, we plan to extend our approach to $|A| > 1$ and consider the insertion of edges incident to all the seeds in A. Moreover, we plan to analyse a generalization of the problem considered in this paper by allowing the deletion of edges incident to seeds. Finally, our intent is to study the same problem in a generalization of ICM, which is the Decreasing Cascade Model. In this model the probability of a node u to influence v is non-increasing as a function of the set of nodes that have previously tried to influence v. From the experimental point of view, our aim is to measure the efficiency of the greedy algorithm in term of expected number of influenced nodes.

References

1. E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an influencer: Quantifying influence on twitter. In *Proc. of the Fourth ACM Int. Conf. on Web Search and Data Mining*, WSDM '11, pages 65–74. ACM, 2011.
2. I. Bogunovic. Robust protection of networks against cascading phenomena, 2012.
3. W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2010.
4. P. Crescenzi, G. D'angelo, L. Severini, and Y. Velaj. Greedily improving our own closeness centrality in a network. *ACM Trans. Knowl. Discov. Data*, 11(1):9:1–9:32, 2016.
5. D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.
6. E. B. Khalil, B. Dilkina, and L. Song. Scalable diffusion-aware optimization of network topology. In *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2014.
7. M. Kimura, K. Saito, and H. Motoda. *Solving the Contamination Minimization Problem on Networks for the Linear Threshold Model.* 2008.
8. M. Kimura, K. Saito, and H. Motoda. Blocking links to minimize contamination spread in a social network. *ACM Trans. Knowl. Discov. Data*, 2009.
9. C. J. Kuhlman, G. Tuli, S. Swarup, M. V. Marathe, and S. Ravi. Blocking simple and complex contagion by edge removal. In *Proc. of the IEEE Int. Conf. on Data Mining (ICDM)*, 2013.
10. G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Math. Program.*, 1978.
11. M. E. J. Newman. Spread of epidemic disease on networks. *Phys. Rev. E*, 66:016128, Jul 2002.
12. M. Papagelis. Refining social graph connectivity via shortcut edge addition. *ACM Trans. Knowl. Discov. Data*, 10(2):12, 2015.

13. D. Sheldon, B. N. Dilkina, A. N. Elmachtoub, R. Finseth, A. Sabharwal, J. Conrad, C. P. Gomes, D. B. Shmoys, W. Allen, O. Amundsen, and W. Vaughan. Maximizing the spread of cascades using network design. *CoRR*, 2012.
14. X. Wu, D. Sheldon, and S. Zilberstein. Efficient algorithms to optimize diffusion processes under the independent cascade model. In *NIPS Work. on Networks in the Social and Information Sciences*, 2015.
15. Y. Zhang, A. Adiga, A. Vullikanti, and B. A. Prakash. Controlling propagation at group scale on networks. In *Proc. of the IEEE Int. Conf. on Data Mining (ICDM)*, 2015.

A Mechanism Design Approach for Allocation of Commodities *

Stefano Bistarelli[1], Rosario Culmone[2], Paolo Giuliodori[2] and Stefano Mugnoz[3]

[1] University of Perugia, Computer Science Department, Italy
[2] University of Camerino, Computer Science Department, Italy
[3] U-Space SRL, Rome - Camerino, Italy

Abstract. We deploy a mechanism design approach for allocating a divisible commodity (electricity in our example) among consumers. We consider each consumer with an associated personal valuation function of the energy resource during a certain time interval. We aim to select the optimal consumption profile for every user avoiding consumption peaks when the total required energy could exceed the energy production. The mechanism will be able to drive users in shifting energy consumptions in different hours of the day. We start by presenting a very basic Vickrey-Clarke-Groves mechanism, we discuss its weakness and propose several more complex variants. This is an extended abstract, for additional details we provide a technical report [1].

1 VCG Mechanisms

Mechanism Design [3–6] is based on the concept of social choice that is simply an aggregation of the preferences of the different participants toward a single collective decision. Mechanism Design attempts to implement desired social choices in a strategic setting, assuming that players act rationally in a game theoretic sense.

Definition 1 (Player's Valuation Function) *Let us consider a set of players* $N = \{1, \ldots, n\}$ *and a set of alternatives or outcomes* A*. Every player* i *has a preference over alternatives that is described by a valuation function:*

$$v_i : A \to \mathbb{R}$$

where $v_i(a)$ *denotes the valuation that player* i *assigns to outcome* a*. Furthermore,* $v_i \in V_i$ *where* $V_i \subseteq \mathbb{R}^{|A|}$ *is a set of possible valuation functions for player* i*.*

* Research partially supported by: project "Viscolla" co-funded by Fondazione Cassa di Risparmio di Perugia, project "BitCoins" co-funded by Banca d'Italia and Cassa di Risparmio di Perugia

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 275–279 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

Definition 2 (Social Choice Function) *The social choice function selects an alternative (or outcome) from the set of alternatives A according to the vector of users' valuation functions:*

$$f : V_1 \times \cdots \times V_n \to A$$

So, an outcome a, from the set A of alternatives, depends on each possible profile $v = (v_1, v_2, \ldots, v_n)$ *:*

$$a = f(v)$$

This outcome is called social choice for that profile.

When considering a mechanism with money, that is a mechanism where there are money transfers between the mechanism and players, a payment function computes money transfers for every players.

Definition 3 (Direct Revelation Mechanism) *A direct*[4] *revelation mechanism* $\mathcal{M}$ *is composed of:*

$$\mathcal{M} = \langle f, p_1, \ldots, p_n \rangle$$

where f is the social choice function with A as the possible outcomes and $p_1, \ldots, p_n$ *are the payment vectors where* $p_i : V_1 \times \cdots \times \ldots V_n \to \mathbb{R}$ *is the amount that user i pays to the mechanism.*

Definition 4 (Utility Function) *Considering a mechanism* $\mathcal{M} = \langle f, p_1, \ldots, p_n \rangle$*, the valuation set* $v = (v_1, \ldots, v_n)$ *and the alternative chosen* $a = f(v_1, \ldots, v_n)$*, the utility for every user i is:*

$$u_i(a) = v_i(a) - p_i(v_i(a), v_{-i}(a)) \tag{1}$$

where v_{-i} *is the (n-1)-dimensional vector in which the i'th coordinate is removed.*

The most famous direct revelation mechanism is the Vickrey-Clarke-Groves (VCG) Mechanism [5].

Definition 5 (VCG Mechanism) *A VCG mechanism determines* $f(v)$*:*

$$f(v) \in argmax_{b \in A} \sum_{j=1}^{n} v_j(b) \tag{2}$$

and $p_i(v)$ *such that:*

$$p_i(v) = h_i(v_{-i}) - \sum_{j \neq i}^{n} v_j(f(v)) \tag{3}$$

for some function $h_1, \ldots, h_n$ *where* $h_i : V_{-i} \to \mathbb{R}$*.*

[4] There exists also the indirect revelation mechanism with money, that differs for the fact that players have private information (player's preferences) and select strategies according to this information set. In this work, we do not consider indirect revelation mechanism because we assume that the users' valuations functions are known.

Now, there are several versions of the model according to the choice of $h_i(v_{-i})$. Important to note that the function h_i can be any arbitrary function but it must not depend on the v_i. One of the most important versions is the VCG mechanism with the Clarke pivot rule, introduced by Clarke [2].

Definition 6 (VCG Mechanism with Clarke Pivot Rule) *A VCG mechanism with Clarke pivot payments determines $h_i(v_i)$:*

$$h_i(v_{-i}) = max_a \sum_{j \neq i}^{n} v_j(a) \tag{4}$$

so $p_i(v)$ becomes:

$$p_i(v) = max_b \sum_{j \neq i}^{n} v_j(b) - \sum_{j \neq i}^{n} v_j(f(v)) \tag{5}$$

where b is the selected alternative if the i-th player is not present in the system[5].

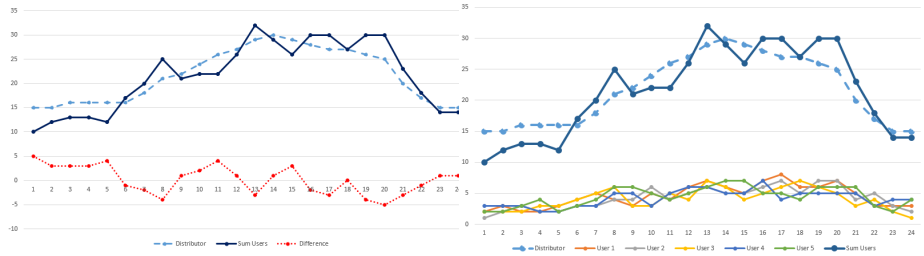
By choosing this kind of $h_i(v_{-i})$ Clarke wants to let the buyer paying only the influence that he has in the system. In fact, an user influences the system when the outcome changes depending on the absence or presence of player i. If the outcome changes significantly when player i is removed, it means that player i strongly affects the system, so he has to pay for his influence (from this concept derives the name “Clarke pivot rule").

2 Problem Formulation and System Model

2.1 Problem Description

In our work, we model an energy allocation problem through a VCG mechanism approach. The main aim is to avoid blackouts when the users' requested energy exceeds the available energy and the energy network must be switched off due to overload. Fig. 1 describes a possible case for one-day time period with trends for the energy functions. The dashed lightblue line represents the distributor's available energy, the continuous darkblue line the energy requested from all consumers. The lines on the bottom represent the single consumption of every user (five users in this case). So in Fig. 1b, we can consider only the consumers "players" and the energy available function as a parameter for the mechanism. The produced energy is a constraint to take into account while maximizing the social welfare. Fig. 1a describes an example of a situation in which energy availability function (dashed lightblue line) and aggregate users' consumption (continuous darkblue line) are compared. Where the set of players contains the distributor

[5] Here, “a" is the optimal assignment including player “i", while “b" is the optimal assignment when we exclude player “i"



(a) Trend of energy availability considering the energy distributor a player.

(b) Trend of energy availability considering only consumers as players

Fig. 1: Comparison between energy functions: users' desired trends, the aggregated desired trend of all of agent and the produced energy function.

and the difference between this two functions (production - consumption, the dotted red line in the graph) must be greater or equal than zero. In fact, another way to tackle this problem is that we can consider that the difference between the production and the aggregate consumption, must be greater or equal than zero for this purpose we can consider the distributor as a player. The main idea is to deploy a mechanism in order to drive users in shifting energy consumptions, according to the produced energy and the consumption preference of the community.

2.2 Model Description

Our objective is to determine a mechanism that will select the optimal energy according to the desired consumption of every player. We were initially inspired by the "Public Project" example provided in [5]. To better understand the idea, we could sum up our problem by drawing connections with the classical knapsack optimization problem. As first solution, we decide to put all into the knapsack only if the volume of the knapsack is sufficient, otherwise we do not put in anything. In the energy case, the distributor will provide energy only if the consumption is not greater than the available energy. For this reason, we consider a scenario with an energy distributor and n energy users ($n+1$ players). The distributor has an amount of available energy and each consumer has a desired consumption, that is represented by a negative value due to the design of the social choice function of the VCG mechanism showed in Eq. 2. This mechanism has the positive aspect that it avoids blackout situations.
As a further solution, we assume that we have a set of object to put into the knapsack minimizing the empty space. We model the scenario removing the distributor from the set of players and considering the available energy a threshold resulting that the energy is provided only to a subset of users with aim to avoiding the waste of produced energy. Moreover, each consumer has a positive desired consumption avoiding a negative net utility introduced in Eq. 1. This

second model with respect to the first one allows to provide energy even if the the aggregated consumption is greater than the production by selecting a subset of users. Regarding the next solution, we assume that we have a liquid instead of a set of objects, so we are able to fill the entire knapsack. So, we introduce a third solution that assigns to users all the available energy till reaching the total amount of produced energy. It means that for each user the mechanism calculates a portion of available energy not greater than his desired consumption. Here as well, the mechanism selects the consumption for a subset of users, however, unlike the second case, there is no wasted energy. The next improved mechanism is similar to the third solution and in addition we introduce the time variable. So, every energy function become a power function over time and the mechanism chooses the energy to be provided according to every user's preferences, allowing the shifting of the consumption in the time interval considering that each user must receive at least his aggregated requested energy.
Our final approach is based on the previous one in which we assign a part of available power thanks to a proportional allocation scheme that provides a positive amount of power to every user.
A final remark is that this model is essentially a game so players must be motivated to play by getting a positive utility. But, in our case study, energy allocation, a user usually has to consume and, consequently, play the game for this reason he can accept also an utility equal to zero.
In this work, we propose several configurations of the mechanism, starting from the simplest to a more complicated configuration which has the property of assigning the available energy to users according to their desired energy minimizing the energy wasting while maximizing the aggregate utility of all users.
A development is to find a different payment scheme that takes into account the actual consumption and energy consumption peaks. The final aim is to stimulate users to behave in a good energy way offering a discount on the electricity bill that will lead to get a positive net utility.

References

1. Bistarelli, S., Culmone, R., Giuliodori, P., Mugnoz, S.: Mechanism Design Approach for Energy Efficiency. ArXiv e-prints (Aug 2016)
2. Clarke, E.H.: Multipart pricing of public goods. Public Choice 11(1), 17–33 (1971), http://dx.doi.org/10.1007/BF01726210
3. Jackson, M.O.: Mechanism theory. In: Derigs, U. (ed.) EOLSS The Encyclopedia of Life Support Systems. EOLSS Publishers: Oxford UK (2003)
4. Narahari, Y.: Game Theory and Mechanism Design, chap. 14. World Scientific Publishing Company Pte. Limited (2014)
5. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory, chap. 9. Cambridge University Press (2007)
6. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, chap. 10. Cambridge University Press, New York, NY, USA (2008)

Merging Frequent Summaries

M. Cafaro, M. Pulimeno

University of Salento, Italy
{massimo.cafaro, marco.pulimeno}@unisalento.it

Abstract. Recently, an algorithm for merging counter-based data summaries which are the output of the *Frequent* algorithm (Frequent summaries) has been proposed by Agarwal et al. In this paper, we present a new algorithm for merging Frequent summaries. Our algorithm is fast and simple to implement, and retains the same computational complexity of the algorithm presented by Agarwal et al. while providing better frequency estimation.

1 Introduction

In 2011, we presented an algorithm [1] for merging in parallel counter-based data summaries which are the output of the *Frequent* [2] algorithm. Recently, we also designed a parallel algorithm for merging Space Saving summaries [3] and an algorithm for mining frequent items in the time fading model [4]. In 2012, a new algorithm for merging counter-based data summaries which are the output of the *Frequent* algorithm has been proposed by Agarwal et al. [5].

Given a data set A of n items t_1, t_2, ..., t_n, the frequency of an item i is $f_i = |\{j \,|t_j = i\}|$. Let $\tilde{f}_i$ be the frequency reported by the algorithm for item i. The absolute error of item i is defined as the difference $|f_i - \tilde{f}_i|$. The (absolute) *total error* is then the sum of the absolute errors related to the items reported by an algorithm.

In this paper, we present a new algorithm for merging Frequent summaries (based on our previous algorithm) which is fast and simple to implement, and retains the same computational complexity of the algorithm presented in [5] while providing better frequency estimation. We briefly recall notations and definitions used in the sequel.

Definition 1. *Given a* multiset $\mathcal{N}$*, with* $|\mathcal{N}| = n$*, and* $2 \leq k \leq n$*, a* frequent item *(or k–majority element) is an element* $x \in N$ *whose* frequency $f_{\mathcal{N}}(x)$ *is such that* $f_{\mathcal{N}}(x) \geq \lfloor \frac{n}{k} \rfloor + 1$*. The frequent items (or k–majority) problem takes as input an array* $\mathcal{N}$ *of n numbers (a multiset), and requires as output the set* $S = \left\{x \in N : f_{\mathcal{N}}(x) \geq \lfloor \frac{n}{k} \rfloor + 1\right\}$

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 280–285 published in CEUR Workshop Proceedins Vol-1720 at http://ceur-ws.org/Vol-1720

Definition 2. *Merged summary.*
Given k, the k-majority parameter, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be respectively the data sets from which the data summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ are derived by an application of the Frequent algorithm, and let $n = |\mathcal{A}_1| + |\mathcal{A}_2|$. The merged summary $\mathcal{M}$ *is the multiset which contains all of the k-majority elements, i.e., all of the elements whose frequency in $\mathcal{A}_1 \biguplus \mathcal{A}_2$ is greater than or equal to $\lfloor \frac{n}{k} \rfloor + 1$. Moreover, all of the guarantees assured by Frequent on its output continue to hold for the summary $\mathcal{M}$ with reference to the input $\mathcal{A}_1 \biguplus \mathcal{A}_2$.*

Definition 3. 2*-way merging problem.*
Input: k, the k-majority parameter; two summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ derived by an application of the Frequent algorithm.
Output: The merged summary $\mathcal{M}$*.*

The paper is organized as follows. We present in Section 3 our algorithm. In Section 4, the proposed algorithm is analyzed in terms of correctness, computational complexity and total error committed. Full details and proofs will appear in a forthcoming extended version. We draw our conclusions in Section 5.

2 Related Work

In [5], Agarwal et al. introduced an algorithm for merging two data summaries $\mathcal{S}_1$ and $\mathcal{S}_2$ outputted by the *Frequent* algorithm. In the following, given a counter C_i, the notation C_i^e refers to the item monitored by the i–th counter, whilst C_i^f refers to its estimated frequency.

Algorithm 1 Merging Algorithm by Agarwal et al.

Require: $\mathcal{S}_1$; an array of counters; $\mathcal{S}_2$; an array of counters; k, k-majority parameter (the number of counters is $k-1$);
Ensure: an array containing k–majority candidate elements

```
1: procedure MERGE(S1, S2, k)                    ▷ a merged summary of S1 and S2
2:     S ← COMBINE(S1, S2);
3:     if S.nz > k − 1 then                      ▷ prune counters in S
4:         for i = k to 2k − 2 do
5:             C_i^f ← C_i^f − C_{k−1}^f;
6:         end for
7:     end if
8:     return S[k ... (2k − 2)];                 ▷ return the last k − 1 counters
9: end procedure
```

The algorithm works as follows. It starts combining as usual the two data summaries, by adding the frequencies of counters monitoring the same item. This could entail, for Frequent summaries, the use of up to $2k-2$ counters in the worst case, when $\mathcal{S}_1$ and $\mathcal{S}_2$ share no item. Let $\mathcal{S}$ be the combined summary, and $\mathcal{S}.nz$ the number of nonzero counters. Moreover, assume, without loss of generality,

that the total number of counters in $\mathcal{S}$, denoted by $\mathcal{S}.length$, is exactly $2k-2$ and they are stored in sorted ascending order. Indeed, it is always possible to pad the first $\mathcal{S}.length - \mathcal{S}.nz$ positions in $\mathcal{S}$ with dummy counters whose frequency is zero.

If $\mathcal{S}.nz \leq k-1$ the algorithm returns the last $k-1$ counters of $\mathcal{S}$. Otherwise, a pruning operation is required. Then, the algorithm subtracts from the last $k-1$ counters the frequency of the C_{k-1}-th counter and returns the pruned counters. The algorithm requires in the worst case time linear in the total number of counters, i.e., $O(k)$ if implemented as described in [5] using an hash table.

We now analyze the total error committed by this algorithm. Clearly, combining the two data summaries can be done without any additional error. However, the pruning operation occurring when the size of $\mathcal{S}$ is greater than $k-1$ induces a total error $E_T = (k-1)C^f_{k-1}$, i.e., $k-1$ times the frequency of the C_{k-1}-th counter in $\mathcal{S}$. The authors proved that the additional error introduced by the merge is within the error bound guaranteed by Frequent.

3 New Merging Algorithm

In this Section we present our algorithm, shown in pseudo-code as Algorithm 2 for merging two Frequent summaries.

Algorithm 2 Merging Algorithm for Frequent summaries.

Require: $\mathcal{S}_1$; an array of counters; $\mathcal{S}_2$; an array of counters; k, k-majority parameter (the number of counters is $k-1$);
Ensure: an array containing k–majority candidate elements

```
1: procedure MERGE(S1, S2, k)                          ▷ a merged summary of S1 and S2
2:     S ← COMBINE(S1, S2);
3:     if S.nz ≤ k − 1 then
4:         return S[k ... (2k − 2)];                    ▷ return the last k − 1 counters
5:     else▷ build the merged summary M, consisting of counters monitoring item e_i
       with frequency f_i, i = 1, ..., k − 1, as follows:
6:         e_1 ← C^e_k
7:         f_1 ← C^f_k − C^f_{k−1};
8:         M[1] ← (e_1, f_1);
9:         for i = 2 to k − 1 do
10:            e_i ← C^e_{k−1+i}
11:            f_i ← C^f_{k−1+i} − C^f_{k−1} + C^f_{i−1};
12:            M[i] ← (e_i, f_i);
13:        end for
14:        return M;
15:    end if
16: end procedure
```

Algorithm 2 starts by combining the two input summaries into a combined summary $\mathcal{S}$. Then, if the number of nonzero counters in $\mathcal{S}$ is less than or equal

to $k-1$, the algorithm returns as merged summary the last $k-1$ counters of $\mathcal{S}$. Otherwise, the last $k-1$ counters are first updated using exact closed-form equations and then reported as output. Actually, these determining equations produce the same merged summary that we would obtain applying the Frequent algorithm to the combined summary $\mathcal{S}$, a procedure we described and proved to be correct in [1]. Indeed, in [1] a slightly modified version of Frequent is used on $\mathcal{S}$, in which the update step is carefully modified so that each update still requires $O(1)$ time in the worst case. These modifications simply consist in one-shot updates: for each item in $\mathcal{S}$ to be processed, we increment one-shot the counter in charge of monitoring it by a number of occurrences equal to the item's counter in $\mathcal{S}$. In the next Section, we shall show the determining equations, state the correctness of the algorithm and analyze its complexity in the worst case and the total error committed. The main result of the paper is the proof that the following properties hold for our algorithm: (i) it retains the same complexity of the Algorithm proposed by Agarwal et al [5], and (ii) its total error committed is smaller or equal.

4 Analysis

4.1 Complexity Analysis

Lemma 1. *The computational complexity of our Algorithm 2 is $O(k)$ in the worst case.*

4.2 Correctness of Algorithm 2

By construction, the combine step producing $\mathcal{S}$ preserves the frequent items in $S_1 \biguplus S_2$ since no item is discarded and no occurrences are lost. Therefore, it suffices to show that our closed-form equations produce the same merged summary which would be outputted by an application of Frequent (the one-shot update version) to the combined summary. Let $\mathcal{S}.length = 2k-2$ and assume $k \leq \mathcal{S}.nz \leq 2k-2$. We denote by C_j the j–th counter in $\mathcal{S}$, $j = 1, \ldots, 2k-2$, and by e_j^i and m_j^i, respectively, the item monitored by the j–th counter of Frequent (denoted as M_j) and its value at the end of the i–th update step, $i = 0, \ldots, k-1$ and $j = 1, \ldots, k-1$. We define $e_j^0 = C_j^e$ and $m_j^0 = C_j^f$, $j = 1, \ldots, k-1$. Indeed, the step zero reflects the situation in which we have already filled the first $k-1$ counters in the Frequent data structure with the corresponding initial $k-1$ counters in $\mathcal{S}$. This is correct owing to the following facts: (i) the counters in $\mathcal{S}$ are stored in ascending sorted order with respect to the frequencies, (ii) the items in $\mathcal{S}$ are distinct and (iii) Frequent works by assigning an item which is not currently monitored to a new counter if available and maintaining the ascending sorted order with respect to the frequencies.

Theorem 1. *For each update step $i = 1, \ldots, k-1$ and position $j = 1, \ldots, k-1$, the values e_j^i and m_j^i can be defined as follows:*

$$e_j^i = C_{i+j}^e \; j = 1, \ldots, k-1 \tag{1}$$

$$m_j^i = \begin{cases} C_{i+j}^f - C_i^f & j = 1, \ldots, k-i \\ C_{i+j}^f - C_i^f + C_{i+j-k}^f & j = k-i+1, \ldots, k-1 \end{cases} \tag{2}$$

4.3 Total Error Committed By Algorithm 2

In what follows, we assume that after the combine step we are left with a data summary $\mathcal{S}$ consisting of more than $k-1$ nonzero counters. Otherwise, both algorithms do not commit any additional error, owing to the fact that the combine step obviously does not incur any error. Therefore, assuming that $\mathcal{S}$ consists of more than $k-1$ nonzero counters, the total error committed by our algorithm is the total error committed by Frequent when applied to $\mathcal{S}$. The counters' frequencies at the end of the $(k-1)$–th update step are m_j^{k-1}, $j = 1, \ldots, k-1$. Consequently, since Frequent underestimates the frequencies, the total error committed is

$$E_T = \sum_{j=1}^{k-1} C_{k-1+j}^f - m_j^{k-1} \tag{3}$$

We claim that the total error committed by Algorithm 2 is less than or equal to the total error committed by algorithm [5].

Theorem 2. *The following inequality holds*

$$\sum_{j=1}^{k-1} (C_{k-1+j}^f - m_j^{k-1}) \leq (k-1) C_{k-1}^f. \tag{4}$$

5 Conclusions

In this paper we have introduced a new algorithm for merging Frequent summaries and compared it to the algorithm proposed by Agarwal et al. from a theoretical perspective. Our algorithm uses exact closed-form equations for determining the outputs; we have shown that it retains the same computational complexity, whilst providing better frequency estimation. Future work includes designing and carrying out several numerical experiments in order to compare the two algorithms we have discussed from a quantitative perspective.

References

1. Cafaro, M., Tempesta, P.: Finding frequent items in parallel. Concurr. Comput. : Pract. Exper. **23** (2011) 1774–1788
2. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: ESA. (2002) 348–360
3. Cafaro, M., Pulimeno, M., Tempesta, P.: A parallel space saving algorithm for frequent items and the hurwitz zeta distribution. Information Sciences **329** (2016) 1 – 19

4. Cafaro, M., Pulimeno, M., Epicoco, I., Aloisio, G.: Mining frequent items in the time fading model. Information Sciences **370 - 371** (2016) 221 – 238
5. Agarwal, P.K., Cormode, G., Huang, Z., Phillips, J., Wei, Z., Yi, K.: Mergeable summaries. In: Proceedings of the 31st Symposium on Principles of Database Systems. PODS '12, New York, NY, USA, ACM (2012) 23–34

On Maximal Chain Subgraphs and Covers of Bipartite Graphs

Tiziana Calamoneri[1], Mattia Gastaldello[1,2], Arnaud Mary[2], Marie-France Sagot[2], and Blerina Sinaimeri[2]

[1] Sapienza University of Rome
via Salaria 113, 00198 Roma, Italy.
[2] INRIA and Université de Lyon
Université Lyon 1, LBBE, CNRS UMR558, France.

Abstract. In this paper, we address three related problems. One is the enumeration of all the maximal *edge-induced* chain subgraphs of a bipartite graph. We give bounds on their number and use them to establish the input-sensitive complexity of the enumeration problem. The second problem we treat is the *minimum* chain subgraph cover. Finally, we approach the problem of enumerating all *minimal* chain subgraph covers and show that it can be solved in quasi-polynomial time.
Keywords: Chain Subgraph Cover Problem, Enumeration Algorithms, Exact exponential algorithms.

1 Introduction

Enumerating (listing) the subgraphs of a given graph plays an important role in analysing its structural properties. Thus, it is a central issue in many areas, notably in data mining and computational biology.

In this paper, we address the problem of enumerating without repetitions all maximal *edge-induced* chain subgraphs of a bipartite graph. These are graphs that do not contain a $2K_2$ as induced subgraph (*i.e.* there are no independent edge sets of size 2). From now on, we will refer to them as *chain subgraphs* for short when there is no ambiguity.

Bipartite graphs arise naturally in many applications, such as biology as will be mentioned later in the introduction, since they enable to model the relations between two different classes of objects. The problem of enumerating in bipartite graphs all subgraphs with certain properties has thus already been considered in the literature. These concern for instance maximal bicliques for which polynomial delay enumeration algorithms in bipartite [6,11] as well as in general graphs [5,11] were provided. In the case of maximal *induced* chain subgraphs, their enumeration

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 286–291 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

can be done in output polynomial time as it can be reduced to the enumeration of a particular case of the minimal hitting set problem [7] (where the sets in the family have cardinality 4). However, the existence of a polynomial delay algorithm for this problem remains open. To the best of our knowledge, nothing is known so far about the problem of enumerating maximal *edge-induced* chain subgraphs in bipartite graphs.

In this paper, we propose a polynomial delay algorithm to enumerate all maximal chain subgraphs of a bipartite graph. We also provide an analysis of the time complexity of this algorithm in terms of input size. In order to do this, we prove some upper bounds on the maximum number of maximal chain subgraphs of a bipartite graph G with n nodes and m edges. This is also of intrinsic interest as combinatorial bounds on the maximum number of specific subgraphs in a graph are difficult to obtain and have received a lot of attention (see for *e.g.* [8,12]).

We then address a second related problem called the *minimum chain subgraph cover* problem. This asks to determine, for a given graph G, the minimum number of chain subgraphs that cover all the edges of G. This has already been investigated in the literature as it is related to other well-known problems such as maximum induced matching (see *e.g.* [3,4]). For bipartite graphs, the problem was shown to be NP-hard [14].

We provide an exact exponential algorithm which runs in time $O^*((2+\varepsilon)^m)$, for every $\varepsilon > 0$ (by O^* we denote standard big O notation but omitting polynomial factors). Notice that, since a chain subgraph cover is a family of subsets of edges, the existence of an algorithm whose complexity is close to 2^m is not obvious. Indeed, the basic search space would have size 2^{2^m}, which corresponds to all families of subsets of edges of a graph on m edges.

Finally, we approach the problem of enumerating all minimal covers by chain subgraphs. To this purpose, we provide a quasi-polynomial time algorithm to enumerate all *minimal* covers by maximal chain subgraphs of a bipartite graph. To do so, we prove that this can be polynomially reduced to the enumeration of the minimal set covers of a hypergraph.

Besides their theoretical interest, the problems of finding one minimum chain subgraph cover and of enumerating all such covers have also a direct application in biology. Nor *et al.* [13] showed that a minimum chain subgraph cover of such a bipartite graph provides a good model for identifying the minimum genetic architecture enabling to explain one type of manipulation, called *cytoplasmic incompatibility*, by bacteria of a genus called *Wolbachia* of their insect hosts. Moreover, as different minimum covers may correspond to solutions that differ in terms of their biological interpretation, the capacity to enumerate all such minimum chain covers becomes crucial.

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the standard graph terminology, as contained for instance in [2]. We consider finite undirected graphs without loops or multiple edges.

Given a bipartite graph $G = (U \cup W, E)$ and a node $u \in U$, we denote by $N_G(u)$ the set of nodes adjacent to u in G and by $E_G(u)$ the *set of edges incident to u in G*. Moreover, given $U' \subseteq U$ and $W' \subseteq W$, we denote by $G[U', W']$ the *subgraph of G induced by* $U' \cup W'$. A node $u \in U$ such that $N_G(u) = W$ is called a *universal node*.

For a *chain graph*, an equivalent condition of not containing a $2K_2$ as an induced subgraph it is that for each two nodes v_1 and v_2 both in U (resp. in W), it holds that either $N_G(v_1) \subseteq N_G(v_2)$ or $N_G(v_2) \subseteq N_G(v_1)$. Given a chain subgraph $C = (X \cup Y, F)$ of G, with the *largest neighbourhood of C*, we mean the neighbourhood of a node x in X for which the set $N_C(x) \subseteq Y$ has maximum cardinality. A set $Y' \subseteq Y$ is a *maximal neighborhood* of G, if there exists $u \in U$ such that $N_G(u) = V'$ and there does not exist a node $u' \in U$ such that $N_G(u) \subset N_G(U')$.

In this paper, we always consider *edge-induced* chain subgraphs of a graph G. Hence, we identify a chain subgraph C of G by its set of edges $E(C) \subseteq E(G)$ and in that case its set of nodes will be constituted by all the nodes of G incident to at least one edge in C (sometimes abusing notation, we more simply write $C \subseteq G$ or $e \in C$). A *maximal chain subgraph C* of a given bipartite graph G is a connected chain subgraph such that no superset of $E(C)$ is a chain subgraph. We denote by $\mathcal{C}(G)$ the set of all maximal chain subgraphs in G.

A set of chain subgraphs $C_1, \ldots, C_k$ is a *cover* for G if $\cup_{1 \leq i \leq k} E(C_i) = E(G)$. Observe that, given any cover of G by chain subgraphs $C = \{C_1, \ldots C_k\}$, there exists another cover of same size $C' = \{C'_1, \ldots C'_k\}$ whose chain subgraphs are all maximal; more precisely, for each $i = 1, \ldots, k$, C'_i is a maximal chain subgraph of G and C'_i admits C_i as subgraph. In order to avoid redundancies, from now on, although not explicitly highlighted, we will restrict our attention to the covers by maximal chain subgraphs.

We denote by $\mathcal{S}(G)$ the set of all minimal chain covers of a bipartite graph G.

An enumeration algorithm is said to be *output polynomial* or *total polynomial* if the total running time is polynomial in the size of the input and the output. It is said to be *polynomial delay* if the time between the output of any one solution and the next one is bounded by a polynomial function of the input size [10].

3 Enumerating All Maximal Chain Subgraphs

The following theorem characterizes the structure of a maximal chain subgraph and it is fundamental for all the other results of the paper.

Theorem 1. *Let $C = (X \cup Y, F)$ be a chain subgraph of $G = (U \cup W, E)$, with $X \subseteq U$, $Y \subseteq W$ and $F \subseteq E$, and let $x \in X$ be a node with largest neighbourhood in C. Then C is a maximal chain subgraph of G if and only if:*

(i) *$N_C(x) = N_G(x)$ is a maximal neighbourhood of G, i.e. there does not exist a node $u' \in U$ such that $N_G(u) \subset N_G(u')$.*
(ii) *$C \setminus E_G(x)$ is a maximal chain subgraph of $G\big[U \setminus \{x\}, N_G(x)\big]$.*

Theorem 1 is the basis of a new recursive algorithm which enumerates all maximal chain subgraphs of G with polynomial delay:

Proposition 1 (Time Complexity and Polynomial Delay). *Let $G = (U \cup W, E)$ be a bipartite graph. It is possible to enumerate all maximal chain subgraphs of G with a total running time of $O(|\mathcal{C}(G)|n^2m)$. Moreover, the solutions are enumerated in polynomial time delay $O(n^2m)$.*

These two statements allow us to achieve some other results briefly described in the following.

3.1 Bounds on the number of maximal chains

By Theorem 1*(ii)*, a maximal chain subgraph can be found by recursively reducing the graph to one whose partition has size $|U|-1$, so we obtain that the maximum number of chain subgraphs is bounded by $\min(|U|, |W|)!$ and that this bound is tight as *e.g.* the *antimatching graph* reach this bound.

We give also a bound on the number of maximal chain subgraphs for a bipartite graph with m edges:

Theorem 2. *Let $G = (U \cup W, E)$ be a bipartite graph with m edges; then $|\mathcal{C}(G)| \leq 2^{\sqrt{m}\log m}$.*

3.2 Minimum Chain Subgraph Cover

Exploiting Proposition 1, the bound obtained in Theorem 2 and the inclusion/exclusion method [1,8] that has already been successfully applied to exact exponential algorithms for many partitioning and covering problems, we are able to provide an $O^*((2+\epsilon)^m)$ algorithm to decide if there exists a chian subgraph cover of size k for a given bipartite observing that the basic search space has size 2^{2^m}.

Theorem 3. *Let $c_k(G)$ be the number of chain subgraph covers of size k of a graph G. Given a bipartite graph G with m edges, for all $k \in \mathbb{N}^*$ and for all $\varepsilon > 0$, $c_k(G)$ can be computed in time $O^*((2+\epsilon)^m)$.*

3.3 Enumeration of Minimal Chain Subgraph Covers

The enumeration of all minimal chain subgraph covers can be polynomially reduced to the enumeration of the minimal set covers of a hypergraph. This reduction implies that there is a quasi-polynomial time algorithm to enumerate all minimal chain subgraph covers. Indeed, the result in [9] implies that all the minimal set covers of a hypergraph can be enumerated in time $N^{\log N}$ where N is the sum of the input size (*i.e.* $n + m$) and of the output size (*i.e.* the number of minimal set covers).

Let $\mathcal{S} = \mathcal{S}(G)$ the set of its minimal chain subgraph covers. Notice that the minimal chain subgraph covers of G are the minimal set covers of the hypergraph $\mathcal{H} := (V, \mathcal{E})$ where $V = E$ and $\mathcal{E} = \mathcal{C}$. Unfortunately, the size of $\mathcal{H}$ might be exponential in the size of G plus the size of $\mathcal{S}$. Indeed not every maximal chain subgraph in $\mathcal{C}$ will necessarily be part of some minimal chain subgraph cover. In order to obtain a quasi-polynomial time algorithm to enumerate all minimal chain subgraph covers, we need to enumerate only those maximal chain subgraphs that belong to a minimal chain subgraph cover.

Given an edge $e \in E$, let $\mathcal{C}_e$ be the set of all maximal chain subgraphs of G containing e and $\mathcal{M}_e$ the set of all edges $e' \in E$ inducing a $2K_2$ in G together with e.

We call an edge $e \in E$ *non-essential* if there exists another edge $e' \in E$ such that $\mathcal{C}_{e'} \subset \mathcal{C}_e$. An edge which is not non-essential is said to be *essential*. Note that for every non-essential edge e, there exists an essential edge e_1 such that $\mathcal{C}_{e_1} \subset \mathcal{C}_e$. Indeed, by applying iteratively the definition of a non-essential edge, we obtain a list of inclusions $\mathcal{C}_e \supset \mathcal{C}_{e_1} \supset \mathcal{C}_{e_2} \ldots$, where no $\mathcal{C}_{e_i}$ is repeated as the inclusions are strict. The last element of the list will correspond to an essential edge.

By the next Lemma we show that it is sufficient to consider the chain subgraphs which contain at least an essential edge.

Lemma 1. *Let C be a maximal chain subgraph of a bipartite graph $G = (U \cup W, E)$. Then C belongs to a minimal chain subgraph cover of G if and only if C contains an essential edge.*

In the following, we show how to detect essential edges.

Theorem 4. *Given a bipartite graph $G = (U \cup W, E)$, for any two edges $e, e' \in E$, $\mathcal{C}_e \subseteq \mathcal{C}_{e'}$ if and only if $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$.*

Notice that, given an edge $e = (u, w) \in E$, $u \in U$ and $w \in W$, it is easy to determine the set $\mathcal{M}_e$, and checking whether $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$ is also easy.

These results allow us to achieve the following result:

Theorem 5. *Given a bipartite graph $G = (U \cup W, E)$, one can enumerate all its minimal chain subgraph covers, i.e. all the elements in $\mathcal{S}$, in time $O(|\mathcal{S}|^{\log(|\mathcal{S}|)+2})$.*

References

1. Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009.
2. Béla Bollobás. *Modern graph theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1998.
3. Andreas Brandstädt, Elaine M Eschen, and R Sritharan. The induced matching and chain subgraph cover problems for convex bipartite graphs. *Theoretical computer science*, 381(1):260–265, 2007.
4. Yu Chang-Wu, Chen Gen-Huey, and Ma Tze-Heng. On the complexity of the k-chain subgraph cover problem. *Theoretical computer science*, 205(1):85–98, 1998.
5. Vânia M.F. Dias, Celina M.H. de Figueiredo, and Jayme L. Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1-3):240 – 248, 2005.
6. Vânia M.F. Dias, Celina M.H. de Figueiredo, and Jayme L. Szwarcfiter. On the generation of bicliques of a graph. *Discrete Applied Mathematics*, 155(14):1826 – 1832, 2007.
7. Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
8. Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2010.
9. Michael L Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.
10. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
11. Kazuhisa Makino and Takeaki Uno. *SWAT 2004, Lecture Notes in Computer Science*, chapter New Algorithms for Enumerating All Maximal Cliques, pages 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
12. J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.
13. Igor Nor, Jan Engelstädter, Olivier Duron, Max Reuter, Marie-France Sagot, and Sylvain Charlat. On the genetic architecture of cytoplasmic incompatibility: inference from phenotypic data. *The American Naturalist*, 182(1):E15–E24, 2013.
14. Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

Author Index