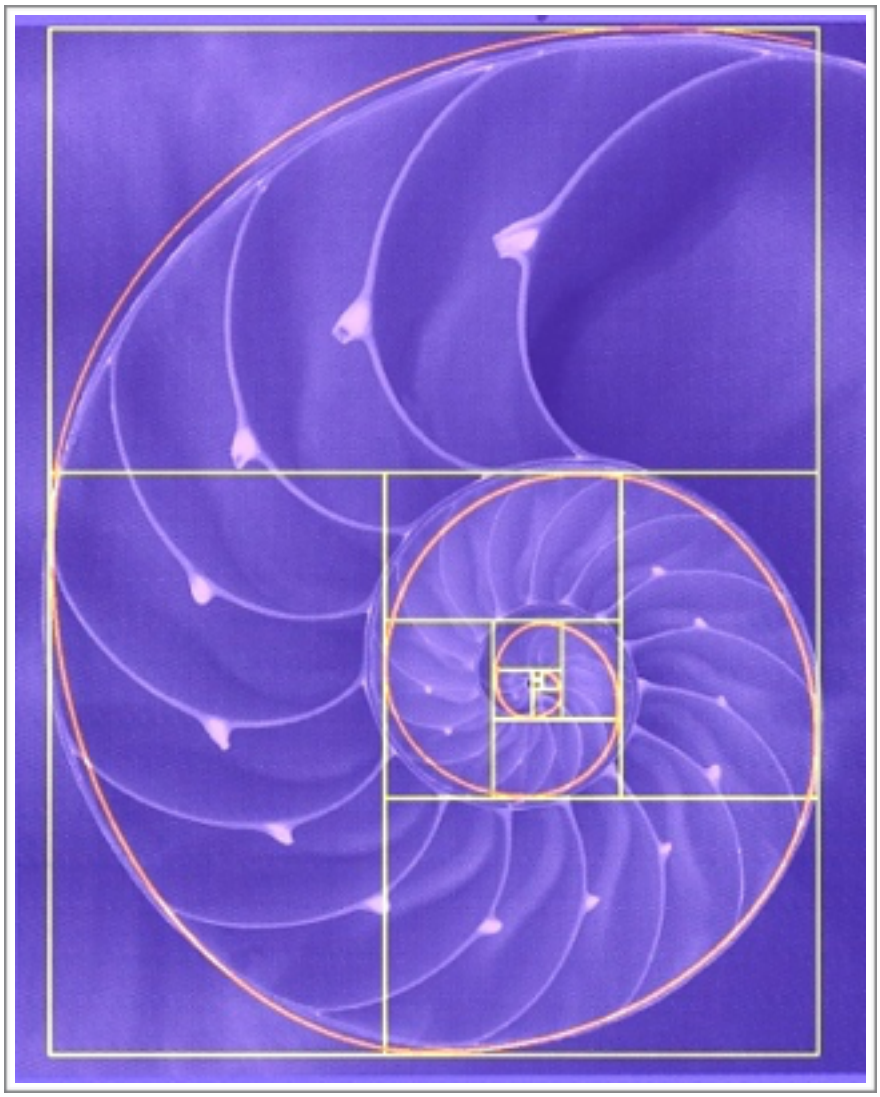
MeMo 2015

2nd International Workshop on Meta Models for Process Languages



Grenoble, June 5, 2015

Part of DisCoTec 2015

Preface

This volume contains the papers presented at the *2nd International Workshop on Meta Models for Process Languages* (MeMo 2015) held on June 5, 2015 in Grenoble. This edition follows the first one, which has been held at DisCoTec 2014 in Berlin.

Metamodels are framework theories aiming to provide general, structural results simplifying and driving the development of models of specific systems and languages. There are frameworks for operational semantics (such as GSOS, graph rewriting systems, Milner's bigraphs, coalgebras), for denotational semantics (such as algebraic/bialgebraic specifications, monads, enriched Lawvere theories, mathematical operational semantics), and for logical semantics (such as metalanguages for deductive systems, i.e. Logical Frameworks). The boundaries between these metamodels are blurred, and techniques and ideas from one can be reapplied to the others.

The goal of the MeMo workshop is to bring together researchers working on and with metamodels, with the aim to share insights, uncover similarities and differences, foster cross-fertilization and stimulate further research. Hence, MeMo solicits contributions in the theory and applications of metamodels: theoretical results, tool implementations, real-world applications, case studies, new application areas, integration of meta-models with programming languages, etc.

At MeMo 2015 we had 5 contributions overall, ranging from structural operational semantics for graph transformation systems, to modal logics for systems with names; from the derivation of labelled transition systems from soft constraint systems, to applications of stochastic bigraphs to the modelization of wireless mesh networks. The program also includes an invited talk by Michele Loreti titled *ULTraS of FuTS*, which are two metamodels for labelled transition systems with quantitative aspects.

I would like to thank the authors for their interest in MeMo, the members of the Program Committee for reviewing the submissions, the Organizing Committee (Søren Debois and Thomas Hildebrandt) and the DisCoTec 2015 organizing committee for their support in the organization of this workshop.

May 15, 2015
Udine

Marino Miculan
MeMo 2015 PC chair

Table of Contents

Program Committee

Patrick Bahr	Department of Computer Science, University of Copenhagen
Iliano Cervesato	Carnegie Mellon University
Vincenzo Ciancia	Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Italy
Søren Debois	IT University of Copenhagen
Fabio Gadducci	Dipartimento di Informatica, Università di Pisa
Tobias Heindel	LFCS, University of Edinburgh
Thomas Hildebrandt	IT University of Copenhagen
Marino Miculan	DiMI, University of Udine
Joachim Parrow	Uppsala University
Jan Rutten	CWI
Pawel Sobocinski	University of Southampton

ULTraS of FuTS

Michele Loreti
Dipartimento di Statistica, Informatica, Applicazioni "G. Parenti"
Università di Firenze
michele.loreti@unifi.it

Labeled transition systems are typically used as behavioural models of concurrent processes. Their labeled transitions define a one-step state-to-state reachability relation. This model can be generalised by modifying the transition relation to associate a *state reachability distribution* with any pair consisting of a *source state* and a *transition label*. Each transition becames a triple of the form $(s,\alpha,\mathscr{P})$. The first and the second components are the source state, s, and the label, α, of the transition, while the third component, $\mathscr{P}$, is the *state reachability distribution*, or *continuation function*, associating a value of a suitable type to each state s'. Values are taken from a preordered set equipped with a minimum that denotes unreachability. By selecting suitable preordered sets, the resulting model can be specialised to capture well-known models of nondeterministic/probabilistic/stochastic processes. For example, in the case of stochastic process algebras the value of the *continuation function* on s' represents the rate of the negative exponential distribution characterising the duration/delay of the action performed to reach state s' from s. This uniform treatment of different behavioural models extends to behavioural equivalences. They can be defined by relying on appropriate measure functions that express the degree of reachability of a set of states when performing multi-step computations. A unifying framework to provide the semantics of process algebras, including their quantitative variants, can be also defined.

Process algebras (see [1] and references therein) have been successfully used in the last thirty years to model and analyse the behaviour of concurrent systems. Apart from specific syntactic operators used to define the term algebra, the basic ingredients of these formalisms are the model called *labeled transition system (LTS)* [13] and *behavioural relations* in the form of equivalences or preorders. By exploiting the structural operational semantic approach [15], an LTS is compositionally associated with each term and behavioural relations over LTS models are introduced to compare process terms describing systems at different levels of abstraction and to investigate properties of interest.

Initially, process algebras were mainly designed to model and assess functional behaviours. However, it was soon noticed that other aspects of concurrent systems are at least as important as the functional ones. Thus, many variants of process algebras have been introduced to take into account quantitative aspects of concurrent systems. There have been proposals of *(deterministically) timed* process algebras, *probabilistic* process algebras, and *stochastic(ally timed)* process algebras, whose semantics have been rendered in terms of richer LTS models quotiented with appropriate behavioural relations.

In order to provide a uniform account of different fully stochastic calculi, in [7], a variant of LTSs, namely *Rate Transition Systems* (RTSs), have been proposed. This model is inspired by the approach taken when modelling probabilistic systems via probabilistic automata, where operators derived from those of the process calculi are applied to probability distributions, as e.g. in [10]. In LTSs, a transition is a triple (P,α,P') where P is the source state, α is the label of the transition, and P' is the target state reached from P via α. On the other hand, in RTSs a transition is a triple of the form $(P,\alpha,\mathscr{P})$, whose first and second component are again the source state and the transition label, but the third component $\mathscr{P}$ is the *continuation function* that associates a real non-negative value with each state P'. A non-zero value represents the rate of the exponential distribution characterising the time needed for the execution

Submitted to:
MEMO 2015

of the action represented by α, necessary to reach P' from P via the transition. Whenever $\mathscr{P}(P') = 0$, P' is not reachable from P via α. RTS continuation functions are equipped with a rich set of operations which make RTSs particularly suitable as a framework for the *compositional* definition of fully stochastic calculi.

In [8] a generalisation RTSs [7], named *State to Function Labeled Transition Systems*, FuTSs for short, has been proposed. FuTSs have generic *commutative semi-rings*, and not just the set of non-negative reals, as co-domain of continuation functions. The semi-ring structure of the co-domain preserves basic properties of primitive operations like sum and multiplication, which prove very useful when modelling composition of rates resulting from (parallel, non-deterministic, sequential) process compositions. Continuation functions are equipped with a rich set of (generic) operations, making FuTSs very well suited as a semantic domain for the *compositional* definition of the operational semantics of process calculi. FuTSs thus support a uniform and systematic understanding of similarities and differences of the many stochastic calculi proposed in the literature [8].

In [3], another step in the direction of a uniform characterisation of the semantics of different process calculi by developing a generalisation of RTS models. The model proposed in [3] is called ULTraS from U*niform* L*abeled* Tra*nsition* S*ystems*. Its transition relation associates with any pair of source state and transition label (s,a) a function $\mathcal{D}$ mapping each possible target state into an element of the support D of a preordered set equipped with a minimum denoted by $\bot_D$. Given a transition $s \xrightarrow{a} \mathcal{D}$, the value of $\mathcal{D}(s')$ expresses the degree of *one-step reachability* of s' from s via that a-transition; if $\mathcal{D}(s') = \bot_D$ then s' is not reachable from s via that a-transition.

The ULTraS model can be used to capture different classes of processes by appropriately choosing D. In particular, we will see that we capture:

1. Fully nondeterministic processes, if D is the support set $\mathbb{B} = \{\bot, \top\}$ of the traditional Boolean algebra.
2. Fully probabilistic processes and processes combining nondeterminism and probability, if $D = \mathbb{R}_{[0,1]}$.
3. Fully stochastic processes and processes combining nondeterminism and stochasticity, if $D = \mathbb{R}_{\geq 0}$.

Modelling state transitions and their annotations is only one of the key ingredients of the description of concurrent processes. One must also combine single transitions into computations and find out ways for determining when two states give rise to behaviourally equivalent computation trees. In [3], the three major approaches to the development of behavioural equivalences are considered and used to define bisimulation, trace, and testing equivalences for the ULTraS model.

An important component of definitions of the three equivalences for ULTraS is a *measure function* $\mathcal{M}_M(s,\alpha,S')$ that returns elements of the support M of another preordered set equipped with a minimum. This function computes the degree of *multi-step reachability* of a set of target states S' from a source state s when performing computations labeled with the sequence of actions α. To capture classical equivalences for the different classes of processes, different measure functions are needed:

1. For nondeterministic processes, the measure of a computation from s to S' labeled with α is $\top$ if the computation exists and $\bot$ otherwise.
2. For probabilistic processes, the measure function yields a value in $\mathbb{R}_{[0,1]}$ that represents the probability of the set of computations from s to S' labeled with α.
3. For stochastic processes, to capture the different equivalences proposed in the literature, one has to distinguish two cases:

- In the *end-to-end* case, given a time threshold $t \in \mathbb{R}_{\geq 0}$, the measure function yields a value in $\mathbb{R}_{[0,1]}$ that represents the probability that the set of computations labeled with α leads from s to S' within t time units.
- In the *step-by-step* case, given a sequence of time thresholds $t_i \in \mathbb{R}_{\geq 0}$, the measure function yields a value in $\mathbb{R}_{[0,1]}$ that represents the probability that the set of computations labeled with α leads from s to S' within t_i time units for each step i.

One of the main objectives of [3] was to asses the different choices that have been presented in the literature in the last twenty years for generalising behavioural equivalences over LTS models to richer models. It is interesting to see which of them are naturally captured by ULTraS and which ones need, instead, an ad hoc treatment. One can then observe that, as long as only models that deal with purely nondeterministic, purely probabilistic, or purely stochastic processes are considered, the known (and generally accepted) equivalences are directly captured. For models that combine probability or stochasticity with nondeterminism, the situation is less straightforward. There are many different ways of interpreting such combinations that influence the way behavioral equivalences are defined, leading to an explosion of potential approaches.

Interestingly enough, for mixed processes ULTraSs lead to new equivalences that were not known in the literature. More precisely, the variant of probabilistic bisimulation equivalence, which has been studied in [5], has a strong connection with PML, the simple probabilistic extension of Hennessy-Milner logic that is in agreement with probabilistic bisimilarity for probabilistic processes without internal nondeterminism [14]. In contrast, the probabilistic bisimulation equivalence in [18] corresponds to a much richer modal logic with a specific operator for capturing probability measures of states reachability [11]. Moreover, the variant of probabilistic testing equivalence, which has been studied in [2, 4], is a conservative extension of the nondeterministic testing equivalence in [6] also when both nondeterministic and probabilistic tests are used, and implies probabilistic trace equivalence also under deterministic schedulers. This is not the case with the probabilistic testing equivalences in [19, 12, 17, 9]. Finally, the variant of probabilistic trace equivalence, which has been studied in [2, 4], is a congruence with respect to parallel composition, while the probabilistic trace equivalence in [16] is not compositional.

References

[1] J.A. Bergstra, A. Ponse & S.A. Smolka (editors) (2001): *Handbook of Process Algebra*. Elsevier.

[2] M. Bernardo, R. De Nicola & M. Loreti (2012): *Revisiting Trace and Testing Equivalences for Nondeterministic and Probabilistic Processes*. In: *Proc. of the 15th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2012)*, *LNCS* 7213, Springer, pp. 195–209.

[3] Marco Bernardo, Rocco De Nicola & Michele Loreti (2013): *A uniform framework for modeling nondeterministic, probabilistic, stochastic, or mixed processes and their behavioral equivalences*. 225, pp. 29–82, doi:10.1016/j.ic.2013.02.004. Available at `http://dx.doi.org/10.1016/j.ic.2013.02.004`.

[4] Marco Bernardo, Rocco De Nicola & Michele Loreti (2014): *Revisiting Trace and Testing Equivalences for Nondeterministic and Probabilistic Processes*. 10, doi:10.2168/LMCS-10(1:16)2014. Available at `http://dx.doi.org/10.2168/LMCS-10(1:16)2014`.

[5] Marco Bernardo, Rocco De Nicola & Michele Loreti (2015): *Revisiting bisimilarity and its modal logic for nondeterministic and probabilistic processes*. *Acta Inf.* 52(1), pp. 61–106, doi:10.1007/s00236-014-0210-1. Available at `http://dx.doi.org/10.1007/s00236-014-0210-1`.

[6] R. De Nicola & M. Hennessy (1984): *Testing Equivalences for Processes*. *Theoretical Computer Science* 34, pp. 83–133.

[7] R. De Nicola, D. Latella, M. Loreti & M. Massink (2009): *Rate-based Transition Systems for Stochastic Process Calculi*. In Albers S., A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas & W. Thomas, editors: *Automata, Languages and Programming. Part II*, *LNCS* 5556, Springer-Verlag, pp. 435–446. ISBN 978-3-642-02929-5.

[8] Rocco De Nicola, Diego Latella, Michele Loreti & Mieke Massink (2013): *A uniform definition of stochastic process calculi*. *ACM Comput. Surv.* 46(1), p. 5, doi:10.1145/2522968.2522973. Available at `http://doi.acm.org/10.1145/2522968.2522973`.

[9] Y. Deng, R.J. van Glabbeek, M. Hennessy & C. Morgan (2008): *Characterising Testing Preorders for Finite Probabilistic Processes*. *Logical Methods in Computer Science* 4(4:4), pp. 1–33.

[10] Y. Deng, R. van Glabbeek, M. Hennessy, C. Morgan, & C. Zhang (2007): *Characterising testing preorders for finite probabilistic processes.* In: *IEEE Symposium on Logic in Computer Science*, IEEE, Computer Society Press, pp. 313–325.

[11] H. Hermanns, A. Parma, R. Segala, B. Wachter & L. Zhang (2011): *Probabilistic Logical Characterization*. *Information and Computation* 209, pp. 154–172.

[12] B. Jonsson & W. Yi (1995): *Compositional Testing Preorders for Probabilistic Processes*. In: *Proc. of the 10th IEEE Symp. on Logic in Computer Science (LICS 1995)*, IEEE-CS Press, pp. 431–441.

[13] R.M. Keller (1976): *Formal Verification of Parallel Programs*. *Communications of the ACM* 19, pp. 371–384.

[14] K.G. Larsen & A. Skou (1991): *Bisimulation Through Probabilistic Testing*. *Information and Computation* 94, pp. 1–28.

[15] G.D. Plotkin (2004): *A Structural Approach to Operational Semantics*. *Journal of Logic and Algebraic Programming* 60/61, pp. 17–139.

[16] R. Segala (1995): *A Compositional Trace-Based Semantics for Probabilistic Automata*. In: *Proc. of the 6th Int. Conf. on Concurrency Theory (CONCUR 1995)*, *LNCS* 962, Springer, pp. 234–248.

[17] R. Segala (1996): *Testing Probabilistic Automata*. In: *Proc. of the 7th Int. Conf. on Concurrency Theory (CONCUR 1996)*, *LNCS* 1119, Springer, pp. 299–314.

[18] R. Segala & N.A. Lynch (1994): *Probabilistic Simulations for Probabilistic Processes*. In: *Proc. of the 5th Int. Conf. on Concurrency Theory (CONCUR 1994)*, *LNCS* 836, Springer, pp. 481–496.

[19] W. Yi & K.G. Larsen (1992): *Testing Probabilistic and Nondeterministic Processes*. In: *Proc. of the 12th Int. Symp. on Protocol Specification, Testing and Verification (PSTV 1992)*, North-Holland, pp. 47–61.

Towards a Channels Allocation Scheme Model for WMNs based on SBRS with Sharing

Rachida Boucebsi Faiza Belala
LIRE Laboratory, Constantine2-Abdelhamid Mehri University, Algeria
rachidaboucebsi@gmail.com faiza.belala@univ-constantine2.dz

The use of formal methods is an effective means to improve the reliability and the capacity of Wireless Mesh Networks. Indeed, the novel technology of WMNs does not deny some limitations imposed by their wireless multi-hop nature. The purpose of this work is to adapt one of these methods to model the WMNs, so that the routing algorithms development, related to this network kind can benefit from it. We propose a formal semantic framework, based on the Stochastic Bigraphical Reactive Systems with sharing (SBRS with sharing), for modeling all entities of the WMN network topology on the one hand, and an efficient and analyzable channel allocation process (CA), in a multi-interfaces/multi-channels environment, on the other hand. Thus, we lean on SBRS model to give a precise and sufficient semantics to WMNs. We gather together the nodes that interfere, exploiting the SBRS features, as the Parent function and the nodes sharing, then we elaborate a set of reaction rules, labeled with rates that express stochastic evolutions of the WMN network while running the CA algorithm. Moreover, our formal model is executed and simulated under the BigraphER environment dedicated to the verification and the simulation of the specified systems.

1 Introduction

Wireless Mesh Networks (WMNs) is an emerging wireless technology which attracts more and more attention of service providers and enterprises, they constitute a very pleasant support for communication due to their flexibility, ease of deployment and minimized costs. Besides, they provide a means of communication for a variety of applications with different service quality requirements in terms of delay, throughput, reliability, confidentiality, etc. In particular, WMNs deployment can cover a wide networks area. They are used to connect multiple LANs wireless; only availability of routers is required. Thus, several wireless technologies support this type of communication, for instance, IEEE802.11 (for Wireless Local Area Networks, WLAN), IEEE802.15.4 (for Wireless Personal Area Networks, WPAN) and IEEE802.16 (for Wireless Metropolitan Area Networks, WMAN). This network type has a mesh architecture where all routers are connected with no central hierarchy. Each node may act as sender, receiver or relay. Thus, every communication between a source node and a destination one is done through multiple hops. WMNs form two-tier architecture, which is mainly compound of Mesh routers, Mesh clients and Mesh Gateways, see Figure 1.

Thanks to independent Mesh routers organization, these entities can form a mesh backbone networks (Backbone). In addition, they maintain the network connectivity and perform the routing process. Generally, Mesh routers are equipped with several radio interfaces for their connection and one interface for the devices connecting. A Mesh router equipped with a bridge Getaway may include several network accesses, such as the Internet network.
Although WMNs have many advantages, several problems persist, such as the security problem, the routing one and the quality of service (QoS), etc. Therefore, this network type does not suffer from

Submitted to:
MeMo 2015

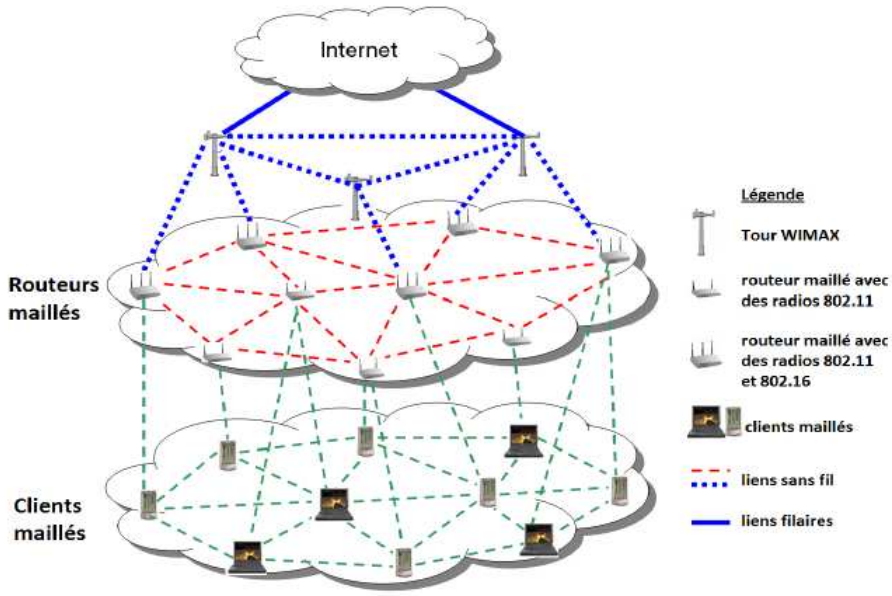


Figure 1: A Generic Architecture of WMN

typical problems often encountered in the other wireless networks as energy consumption for sensor networks and mobility for MANET ones. However, as they are multi-hop wireless networks, they invoke other problem kinds, mainly the interference issue which affects their performances and causes the flow degradation and the node overloading. This phenomenon is undesirable since it affects directly the communications occurring in the same area and on the similar or converged channels.
To tackle this problem, some existing works opt to use multiple radio interfaces and multiple channels on one hand, and duplicate paths between the source and the destination nodes to spread data traffic across multiple paths, on the other hand. But, we should note that two neighboring nodes cannot communicate unless their radios share the same channel. However, the reuse of the same channel in a neighborhood must be limited. Simultaneous transmissions on the same channel may cause collisions and lead to throughput degradation. Indeed, in an interference range, all links using the same channel, cannot transmit at the same time and should not share the same channel capacity.
In order to improve the WMNs capacity, several researches are currently conducted regarding the channel assignment when the routing is performed. We have already proposed in [18] an extended routing process taking the channels allocation in WMNs as a central key for minimizing the interference. Validation and analysis of the most existing approaches are based on simulation and test bed experiments. Although these methods are important and valuable for protocol evaluation, they still remain limited, they are very expensive in term of time consuming and are not exhaustive. Therefore, no general guarantee can be given about protocol behavior for a wide range of unpredictable deployment scenarios.
The use of formal methods is an effective means to improve the reliability and the capacity of Wireless Mesh Networks. They may provide valuable design tools and contribute to the evaluation and the verification of routing protocols. This paper adopts an extension of Bigraphical Reactive Systems (BRS, in short) [13] as a semantic framework for specifying WMNs topology and channel assignment during the routing process. For this purpose, Stochastic Bigraphs with Sharing [21] seem more appropriate than ordinary bigraphs since they are able to define spatial overlapping locations.

In a previous work [19], we have shown that graphical aspect and rigorous basis of ordinary BRS are suitable for representing both locality and connectivity of routers and channels in WMNs during their reconfiguration process. We aim in this paper, to improve our model in order to preserve the WMNs main features, as the multi-hop and the overlapping of channels. We elaborate a Stochastic BRS with sharing model, called BiS-WMN*, to formally describe the network topology and its behavior while dealing with channel allocation during the routing process. Networks and their paths are represented as nodes of the Topology root, and channels with their eventual interferences represent the sharing nodes belonging to another distinctive root (called Interference root). Then, stochastic reaction rules, labeled with rates, define possible evolution and reconfiguration of these WMN entities. Moreover, a bigraphical simulation engine (BigraphER tool [20]) may be applied to execute and simulate our obtained model.
The rest of the paper is organized as follows. In section 2, we present related work. In section 3, we give a brief overview on Bigraphical Reactive Systems (BRS) and their relevant extensions. Section 4 presents our bigraphical specification of WMN networks. In section 5, we show how we encode and execute our bigraphical model with BigraphER tool. Finally, some concluding remarks and ongoing work round up the paper.

2 Related work

Using formal methods in WMNs context is relatively new, but it may have a great benefit and help in this concern. Some formalisms, such as Process Algebra and Petri nets, have been proposed to solve specific problems of WMNs. In [4], authors gave a process algebra based model, called AWN (Process Algebra for Wireless Mesh Networks) to specify WMNs routing protocols, hence it is about the AODV (Ad hoc On Demand Distance Vector) core. This result (AWN) was also used in [7] to prove that the sequence number cannot guarantee the non-existence of loops in protocols. In this context, further works [3], [15], [16] have resumed this formalism to model-check, respectively AODV and AODVv2 routing protocols, but in the context of WMN, with UPPAAL and SMC-UPPAAL tools.
The same routing protocol versions are also formally analyzed by [9] and [11] using Colored Petri Nets. This formalism has also been used by [8] to verify the safety of WMNs, solving some attack kinds, like e.g. "Black hole attack".
The above-cited works address neither the channels allocation nor the multi-path routing in the multi-radio WMNs context. In addition, both types of the used formalisms have been integrated in the Bigraph theory [12]. As a result, we contribute here by providing a generic BRS-based model in order to specify and analyze WMNs topology and their inherent behaviors, while insisting on channels allocation process. Indeed, the adopted formalism (Stochastic Bigraphical Reactive Systems with sharing) seems very appropriate and able to represent both locality and connectivity in WMN networks. It gives a formal meaning to all WMNs entities (signal, nodes, packets, etc.), and it allows us to attribute a rate to the reaction rule in order to control their application during the channel allocation process.

3 Bigraphs presentation

Bigraphical Reactive Systems (BRS) theory was developed by Robin Milner [17] to model and analyze the distributed mobile code. The BRS is simply a bigraph (a graph type) equipped with reaction rules. A bigraph results from two merged graphs, places graph and links graph (Figure 2). Places graph specifies the nodes hierarchy (nested locality of components) while the links graph represents the connections between these nodes.

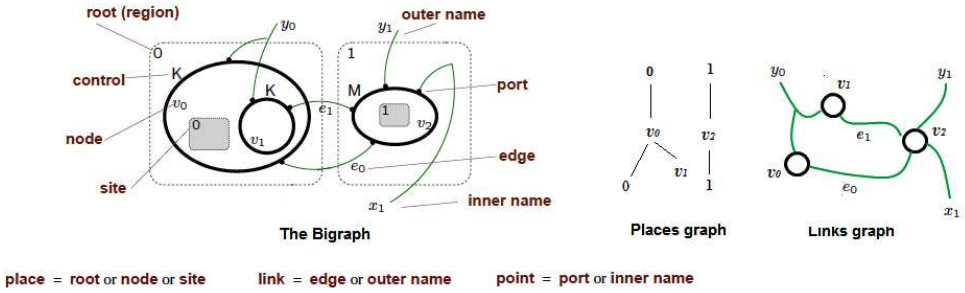


Figure 2: Anatomy of an ordinary bigraph

A bigraph may contain the following elements: roots (regions), nodes, sites, edges, ports, inner and outer names (see Figure 2). Links graph encloses nodes and edges, where each edge connects two or more node ports. Place graphs are contained inside regions and may also contain sites that can be replaced by a region or another bigraph. A node can have ports, seen intuitively as points for link connections. Ports are assigned to nodes by controls which define the arity of a node (the number of ports). A set of controls is called a signature. The wide use of BRS in various application areas has prompted the founders of this theory to expand it in several ways.

- Binding Bigraphs [22] are proposed to introduce the full independence relaxation of placing and linking notions, i.e. some links possess a locality concept in a bigraph. For instance, an edge bounded to a given node can only link ports that lie within this node.
- Directed Bigraphs [5], this extension aims to assign a direction to edges for detecting the resource request flow. For this case, edges are considered as resources and names as resources requests.
- Stochastic BRSs (SBRS) [10] associate rates to rewrite rules allowing deriving the rule activities on a given bigraph. Therefore, the state space generated by a SBRS can be naturally transformed into a Continuous Time Markov Chain (CTMC), so a quantitative analysis can be achieved through tools as the stochastic model checker PRISM.
- Bigraphs with Sharing [21] is a novel generalization of Milners bigraphs in which two or more parents nodes may share the same child ”node”. So, the parent relation forms a DAG, and not a generic graph. therefore, the locality notion is updated. As a consequence, places graph has been changed to support this sharing and the parent mapping is a binary relation instead of a function. This modification is sufficient to allow places to have zero or more parents.

In this paper, we are interested in a combination of the two later extensions of BRS. The Stochastic Bigraphical reactive systems with sharing (SBRS with sharing) have been proposed initially by Muffy Calder and Michele Sevegnani [14], then expanded with priorities to give a spatial order to reaction rules (PSBRS with sharing).

4 Our Approach

In this section, we focus on how the BRS are able to define both static and dynamic aspects of WMNs. First, we need to define the topology (static aspect) and the interference aspects. This later represents the referred problem that we want to minimize by designing a channel allocation process (dynamic aspect)

thanks to the reaction rules. The routers and the channels localities may be defined by Places and Links graphs respectively. The reactions rules model the channels allocation, given a set of links forming the path between a source router and a destination one. We assume that all the routers have the same number (two, 2) of radio interfaces and the channel allocation process concerns only one path, i.e, we are interested here by the intra-flow interference. We propose in what follows the Stochastic BRS-based specification of WMNs, called BiS-WMN* (*Bigraph with Sharing for Wireless Mesh Networks*).

4.1 WMN Topology Model

The general topology of a WMN is defined through one bigraph root (*Topology root*) of the BiS-WMN* model , the second root (*Interference*) and some stochastic reaction rules are designed to support the channel allocation process during routing. Particularly, we give a corresponding rules set (see Table 1) for abstracting the most important elements of a WMN network in the bigraph with sharing syntax. Then, we use this abstraction to associate a formal semantics to a WMN topology consisting of several fixed routers and mobile channels. Each router has two radio interfaces which represent the communication means of a router via its channels. Thus, we identify a bigraph definition consisting of two roots: *Topology* and *Interference*. The first root (see Figure 3) contains the nodes: *Network, Path, RGRi* (to specify a router range), *C0* (represents the signal availability between two routers), *IIP* (Interference-Inside-Path) and a set of routers as links (*ri*), related to *RGRi* nodes. The *Interference root* includes:*IC* nodes (interference for a given channel), each one (*ICi*) nests a set of the *Ci* non-orthogonal channels. This bigraphical structure (see Figure 3), which is independent of the WMN topology, is added to support the specification of the channel allocation process. This latter is mainly based on the CA algorithm proposed in [18]. We resume the IEEE.802.11 standard and reduce the channels number to eight (8) in order to facilitate the readability of the figure. We note that the channel interference in this standard is limited to two posterior/anterior adjacent channels. For example, the channel of number 6 interferes with its two successors channels: 7,8 and its two previous ones: 5,4

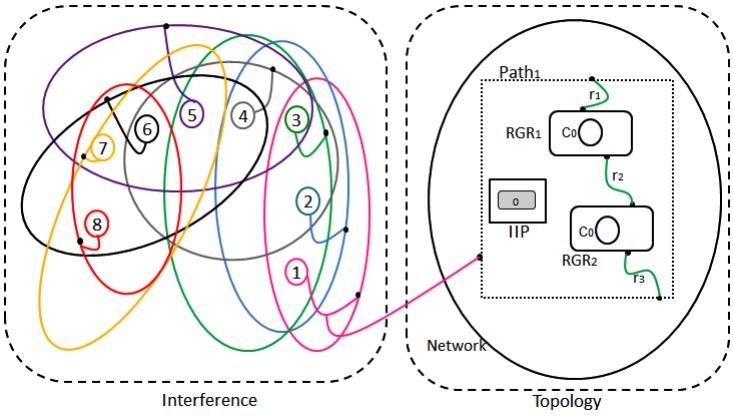


Figure 3: The bigraph of BiS-WMN* model

Places graph (Figure 4) in this case contains two trees that represent the hierarchical nesting of the various nodes, for example, a path (*Path*) is located within a Network node, etc. Note that in the *Interference root* tree, each *ICi* node is attached to a set of channels that interfere with channel *Ci*, i.e., {*Ci-2, Ci-1, Ci+1, Ci+2*}, except for channels end of the standard (*C1* and *C8* in our case). Indeed, the *ICi* node type can share some *Ci* nodes to avoid the nodes duplicating, while the link graph (Figure 5)

shows the possible relationships between these entities. We identify, for instance the links *ri* (routers) between the *RGRi* nodes and *Cil* between *ICi* and *Ci* nodes, etc.

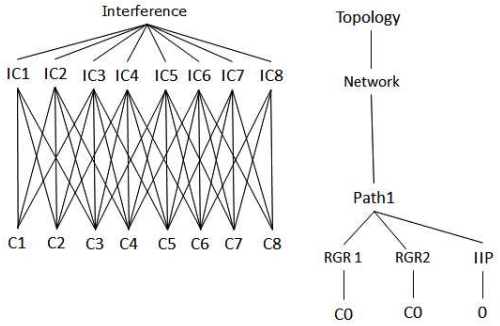


Figure 4: Places graph

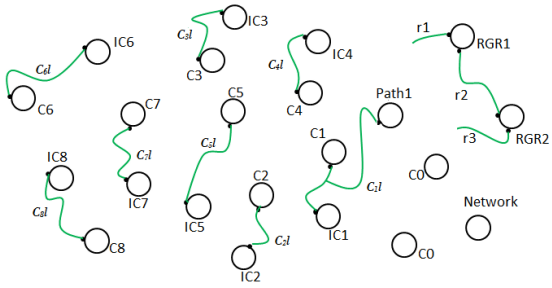


Figure 5: Links graph

4.2 Channel Allocation Process Modeling

Given the above WMN static part modeling based on the bigraph with sharing formalism, to specify a rational Channels Allocation process (CA) which minimize the problem of interference between the used channels, we need to define a set of reaction rules (see last line of Table 1) ensuring the multi-radio WMN network evolution (dynamic part). Specifically, this paper contribution allows decorating each reaction rule by a rate expressing stochastic evolution of the network. Mainly, we conceive the following four meta-rules that will be applied on the bigraph Topology regarding its Interference root.

- Meta-rule 1: the selected channel *Ci* for a given path is initially hosted inside the *RGRi* node replacing the *C0* node. The first reaction rule in Figure 11 illustrates a simple instantiation of this meta-rule where the Topology root contains three routers, and four available channels (Interference root) with possibly interference of one channel (after and before) each one. This rule may have the simplest rate, because at time *t=0*, this is the only relevant rule.
- Meta-rule 2: allows pursuing this type of channels assignment (*Meta-rule1*) while achieving the interference verification (*Meta Rule3*). The second reaction rule in Figure 11 illustrates its instantiation in the same example. The rate of this rule is calculated on the basis of the channels frequencies, i.e., the current channel frequency (fx) and a given available channel frequency (fy) : $\rho(fx,fy) = (fy-fx)/p$. Where p is the interference interval defined between channels (in our case p =2).
- Meta-rule 3: checks if the interference between channels of a given path exists, thus the *IIP* node receives both concerned nodes *RGRi* and *RGRj* as indicated in the third reaction rule of Figure 11. Then, the rate of this rule is: $\rho(fx,fy) = (fy)/(p+fx)$.
- Meta-rule 4: must be applied when the channel interference problem is caused, it replaces the latest allocated channel by its successor, and it transits IIP node to its first state as in the fourth reaction rule of Figure 11. The rule rate is defined as follows: $\rho(fx,fy) = (fy)/(p+fx)$.

The use of rates guides the evolution process avoiding the use of all the declared rules. In the final step of this formalization approach, we summarize our modeling approach by defining a formal meaning to all

WMN Concepts	**Bigraph-based Abstract Syntax**
Topology	Topology root
Interference definition	Interference root
Range	Nodes $RGRi \in V_{BiS\text{-}WMN}$, such that : $Prnt_{BiS\text{-}WMN}(RGRi)=Pathj \cup Network$
Path	Nodes $Pathi \in V_{BiS\text{-}WMN}$, such that : $Prnt_{BiS\text{-}WMN}(Pathi) = Network$
Interference inside path	Node $IIP \in V_{BiS\text{-}WMN}$, such : $Prnt_{BiS\text{-}WMN}(IIP)= Path$
Channel	Nodes $Ci \in V_{BiS\text{-}WMN}$, such that : $Prnt_{BiS\text{-}WMN}(Ci) = ICi \cup RGRi$
Network	Nodes $Network \in V_{BiS\text{-}WMN}$, such that : $Prnt_{BiS\text{-}WMN}(Network) = Topology\ root$
Channels overlapping	Nodes $ICi \in V_{BiS\text{-}WMN}$, such that : $Prnt_{BiS\text{-}WMN}(ICi)= Interference\ root$
Router	Link $ri \in E_{BiS\text{-}WMN}$
Relationship between a channel *Ci* and the interfering channels set or also the Path in which it will be attached	Link $Cil \in E_{BiS\text{-}WMN}$
Hosting a channel location inside a range area	Node $C0 \in V_{BiS\text{-}WMN}$, such : $Prnt_{BiS\text{-}WMN}(C0)= RGRi$
Channel Allocation process	The reaction rules set that defines the deployment of a node *Ci* belonging to the *Interference root* inside a node *RGRj* of the *Topology root*

Table 1: Mapping WMN concepts to Stochastic BRS with sharing Elements

WMN entities (see Table1) and their possible reconfigured states when dealing with Channels Allocation algorithm (the four meta-rules application). The entire semantics assigned to a WMN network in the case of our study (CA) is given by the following definition.

Definition 1: We associate a Stochastic BRS with sharing based model, ***BiS-WMN**** to each WMN network in evolution, consisting of a bigraph ***BS*** and a set of reaction rules ***SR***. Where,

$\boldsymbol{BS}$= ($\boldsymbol{V_{BiS\text{-}WMN}}$,$\boldsymbol{E_{BiS\text{-}WMN}}$,$\boldsymbol{ctrl_{BiS\text{-}WMN}}$,$\boldsymbol{GP_{BiS\text{-}WMN}}$,$\boldsymbol{GL_{BiS\text{-}WMN}}$,$\boldsymbol{PT_{BiS\text{-}WMN}}$) : $\boldsymbol{(m,\ \phi) \rightarrow (2,\ \phi)}$, and ***SR*** is a set of stochastic reaction rules obtained by instantiating the four meta-rules defined below.

- $V_{BiS\text{-}WMN}$ = {*RGR, C, IC, C0, Network, Path, IIP*}, it includes all node types of the WMN network bigraph.
- $E_{BiS\text{-}WMN}$ = { *ri , Cil*}, is a set of links; where *ri* represents a router, *Cil* is a link connecting a channel *Ci* with its *ICi* interference node, or with the Path node if this channel is selected as the first channel in the concerned path.
- $ctrl_{BiS\text{-}WMN}$: $V_{BiS\text{-}WMN} \rightarrow K$, assigns kinds (types) to all nodes, K = {*Ci (1, atomic), C0 (0, atomic), ICi: (1, composite), RGRi: (2, composite), Network: (0, composite), Path: (1, composite), IIP (0, composite)*}.

- $GP_{BiS\text{-}WMN}$ et $GL_{BiS\text{-}WMN}$, are respectively places and links graphs where : $GP_{BiS\text{-}WMN}$= ($V_{BiS\text{-}WMN}$, $ctrl_{BiS\text{-}WMN}$, $prnt_{BiS\text{-}WMN}$) : $\boldsymbol{m} \rightarrow \boldsymbol{2}$. $GL_{BiS\text{-}WMN}$= $V_{BiS\text{-}WMN}$,$E_{BiS\text{-}WMN}$,$ctrl_{BiS\text{-}WMN}$, $link_{BiS\text{-}WMN}$): $\phi \rightarrow \phi$. In our case $X_{BiS\text{-}WMN}$ and $Y_{BiS\text{-}WMN}$ are empty, it is about a closed bigraph.
- $link_{BiS\text{-}WMN}$: $P_{BiS\text{-}WMN} \rightarrow E_{BiS\text{-}WMN}$ is an application that shows the data flow from the ports set: $P_{BiS\text{-}WMN}$ to the edges set: $E_{BiS\text{-}WMN}$.
- $prnt_{BiS\text{-}WMN}$ = {*(Topology, Network), (Network, Path), (Network, RGRi), (Path, RGRj), (Path, IIP), (IIP, RGRi), (Interference, ICk), (ICk, Ck), (RGRi , C0), (RGR, Ci)*}, is a binary relation which associates each node to its hierarchical parents.
- $P_{BiS\text{-}WMN} = PC \cup PIC \cup PPT \cup PRG \cup PIIP$ avec $PC = \{ICiP\}$, $PIC = \{CiP\}$, $PPT = \{CP\}$, $PIIP = \{IIPP\}$,$PRG = \{R1iP, R2jP\}$, thus we associate one type for each port.

We note that: *(m, ϕ) and (2,ϕ)* define the bigraph interfaces, where *m* is the number of the used sites, in our case it is equal to the number of nodes *IIP*. It is obvious to note also that the regions number is fixed to two (2).

5 BiS-WMN* Model Execution

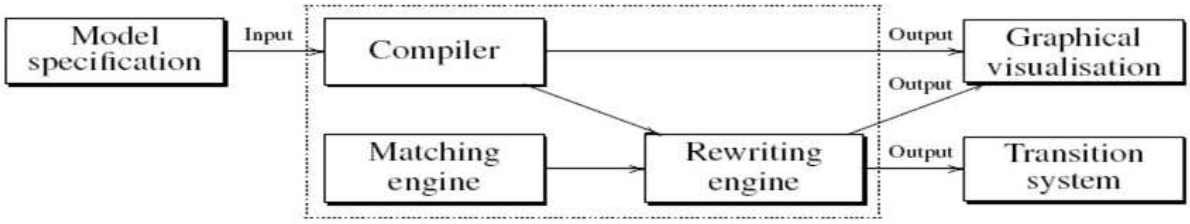


Figure 6: Architecture of the command-line tool. The modules are represented by the boxes within the dotted box. Unlabeled arrows show the dependency relation between the modules.

```
fun ctrl C(f) = 0;      ctrl IC1 = 1;      ctrl RGR1=2;
fun ctrl C1(f) = 1;     ctrl IC2 = 1;      ctrl RGR2=2;
fun ctrl C2(f)= 1;      ctrl IC3 = 1;      ctrl Network = 0;
fun ctrl C3(f) = 1;     ctrl IC4 = 1;      ctrl Path = 1;
fun ctrl C4(f)= 1;      ctrl IC5 = 1;      ctrl IIP = 0;
fun ctrl C5(f) = 1;     ctrl IC6 = 1;
fun ctrl C6(f) = 1;     ctrl IC7 = 1;
fun ctrl C7(f) = 1;      ctrl IC8 = 1;
fun ctrl C8(f) = 1;
```

Figure 7: Bigraph signature declaration

A set of practical tools are developed around BRS models in order to manipulate, execute and analyze the bigraphical systems. The most known ones are: BPL Tool [1], Big Red [2] and BigMc [6]. BigraphER (Bigraph Evaluator and Rewriting) [20] is a recent tool that implements BRS-based model and Stochastic BRS-based model too, it supports places graph with sharing. BigraphER is composed of

```
fun sreact ac2 (a,f,p)=  Network.Path{l1}.(RGR1{r1,r2}.C(f) | RGR2{r2,r3}.C(a)|IIP.1) ||
                         share (C1(f){l1} || C2(f+1){l2} || C3(f+2){l3} || C4(f+3){l4}|| C5(f+4){l5}
                         || C6(f+5){l6}|| C7(f+6){l7}|| C8(f+7){l8})
                         by ([{0,1,2}, {0,1,2,3},{0,1,2,3,4},{1,2,3,4,5},{2,3,4,5,6},
                         {3,4,5,6,7},{4,5,6,7},{5,6,7}],8)
                         in (id{l1,l2,l3,l4, l5, l6, l7, l8} | IC1{l1} | IC2{l2} |
                         IC3{l3}| IC4{l4}| IC5{l5}| IC6{l6}| IC7{l7}| IC8{l8})
                         ->
                          Network.Path{l1}.(RGR1{r1,r2}.C(f) | RGR2{r2,r3}.C(f+1)|IIP.1) ||
                         share (C1(f){l1} || C2(f+1){l2} || C3(f+2){l3} || C4(f+3){l4}|| C5(f+4){l5}
                         || C6(f+5){l6}|| C7(f+6){l7}|| C8(f+7){l8})
                         by ([{0,1,2}, {0,1,2,3},{0,1,2,3,4},{1,2,3,4,5},{2,3,4,5,6},
                         {3,4,5,6,7},{4,5,6,7},{5,6,7}],8)
                         in (id{l1,l2,l3,l4, l5, l6, l7, l8} | IC1{l1} | IC2{l2} |
                         IC3{l3}| IC4{l4}| IC5{l5}| IC6{l6}| IC7{l7}| IC8{l8})
                         @ (((f+1)-f)/p)    ;
```

Figure 8: Stochastic reaction rule

```
fun big n_0(a,f)= Network.Path{l1}.(RGR1{r1,r2}.C(a) | RGR2{r2,r3}.C(a)|IIP.1) ||
                  share (C1(f){l1} || C2(f+1){l2} || C3(f+2){l3} || C4(f+3){l4}|| C5(f+4){l5}
                  || C6(f+5){l6}|| C7(f+6){l7}|| C8(f+7){l8})
                  by ([{0,1,2}, {0,1,2,3},{0,1,2,3,4},{1,2,3,4,5},{2,3,4,5,6},
                  {3,4,5,6,7},{4,5,6,7},{5,6,7}],8)
                  in (id{l1,l2,l3,l4, l5, l6, l7, l8} | IC1{l1} | IC2{l2} |
                  IC3{l3}| IC4{l4}| IC5{l5}| IC6{l6}| IC7{l7}| IC8{l8});
```

Figure 9: Initial bigraph

an OCaml library and a command-line tool. This later uses data structures that OCaml library provides for their programming interfaces. Command-line tool (Figure 6) [21] contains a compiler, a matching engine and a rewriting engine. The tool input is a model specification written in BigraphER specification language. However, the output can be textual or a graphical representation of the bigraph. Moreover, the tool may offer a graphical representation of each state. This result is obtained thanks to rewriting engine that builds a graph by iteratively applying the reaction rules to each state.

```
sbrs
int f = { 1, 2, 3, 4, 5, 6, 7, 8 };
int a={0};
int p={2};

  init n_0(0,1);

  rules = [
  {ac1(a,f),ac2 (a,f,p),ac22 (a,f,p),ac23 (a,f,p),verif2(f,p),verif23(f,p),maj2(f,p), maj23(f,p)}

     ];
endsbrs
```

Figure 10: SBRS Declaration Example

We have transcribed our model (BiS-WMN*), through the generic example of (Figure 3), in the BigraphER specification language to execute it, and simulate its system behavior. Our specification is divided into four parts. The first defines the signature of the model; it is illustrated in (Figure 7) where we declare the controls of the different used nodes. Next, we specify a set of stochastic reactions rules for

this example; we especially instantiate those presented above. A rule example that handles an allocation of a given channel (C2 in our case) is presented in (Figure 8). An algebraic specification of our initial bigraph must also be given (see Figure 9). Finally, a complete definition of the corresponding reactive system is indicated at (Figure 10)(for this example) where we identify the initial bigraph (n_0) and the set of considered reaction rules.

The BigraphER tool performs the execution of this example. As important result, we affirm that not all defined reaction rules are applied and we have three possible states. Table 2 shows the result of our reaction rules execution, each BiS-WMN* state is represented graphically as a combination of the two (places and links) graphs. The first state is the initial bigraph, the second and the third ones are obtained after applying the stochastic rules: Meta rule1 and Meta rule2. We notice that only channel allocation reaction rules are applied. Indeed, when calculating rule rates, we find that channel numbered 4 is the most appropriate one after channel numbered 1 (i.e., it has the greater rate). Therefore, we have no need to apply verification (Meta-rule3) or updating (Meta-rule4) stochastic reaction rules in this particular example.

6 Conclusion

We have presented in this work a semantic framework for modeling multi-radio wireless mesh networks. Specifically, we have shown the interest of a given BRS extension, coupling the bigraph with nodes sharing and stochastic bigraph, to support the modeling of WMNs.
We have defined a complete bigraphical model, called BiS-WMN*, for both the static and the dynamic parts of a WMN network. A generic topology (structure) of this network is represented thanks to a *Topology root* of the bigraph. While its reconfigured states are specified, given another *Interference root*, by some stochastic reaction rules. Each rule is decorated with a rate expressing a possible stochastic evolution of the WMN network while running the CA algorithm of [18]. Besides, our formal model was executed and simulated under the BigraphER tool.
This work provides a new opportunity to formal specify and verify the WMN routing protocols. Since our bigraphical model is generic, it can be extended by several structures regarding Wireless Mesh Networks features. In the near future, we plan to use the added stochastic information, in order to express and verify the non-functional properties (quality of service) and thus, ensure the flow improvement and performance of WMNs networks.

References

[1] L. Birkedal E. Hjsgaard A.J. Glenstrup, T.C. Damgaard (2008): *An implementation of bigraph matching*.

[2] Thomas T. Hildebrandt Alexander Faithfull, Gian Perrone (2012): *BigRed: a development environment for bigraph. 4th International Workshop on Graph Computation Models.*

[3] Peter Hoefner Annabelle McIver Marius Portmann Wee Lum Tan Ansgar Fehnker, Robert van Glabbeek (2012): *Automated Analysis of AODV using UPPAAL. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems.*

[4] Peter Hoefner Annabelle McIver Marius Portmann Wee Lum Tan Ansgar Fehnker, Robert van Glabbeek (2012): *A Process Algebra for Wireless Mesh Networks. 22nd European Symposium on Programming.*

[5] M. Miculan D. Grohmann (2007): *Directed bigraphs. Electronic Notes in Theoretical Computer Science.*

[6] T. Hildebrandt G. Perrone (2012): *A Model Checker for Bigraphs. Proceedings of the 27th ACM Sym. in Applied Computing ACM-SAC'12.*

[7] Wee Lum Tan Marius Portmann Robert van Glabbeek, Peter Hoefner (2013): *Robert van Glabbeek, Peter Hoefner, Wee Lum Tan and Marius Portmann*. 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 1-10, Barcelona, Spain, November.

[8] Qiang Zhou Hejiao Huang (2012): *Petri-net-based Modeling and Resolving of Black Hole Attack in WMN*. IEEE 36th International Conference on Computer Software and Applications Workshops.

[9] C. Yuan J. Billington (2009): *On modeling and analyzing the dynamic MANET on-demand (DYMO) routing protocol*. Transactions on Petri Nets and Other Models of Concurrency III., ser. Lecture Notes in Computer Science. Springer Berlin /Heidelberg., vol. 5800.

[10] A. Troina J. Krivine, R. Milner (2008): *Stochastic bigraphs*. Electronic Notes in Theoretical Computer Science,218:7396.

[11] L. Kristensen K. Espensen, M. Kjeldsen (2008): *Modeling and initial validation of the DYMO routing protocol for mobile ad-hoc networks*. Applications and Theory of Petri Nets (Petri NETS08), ser. LNCS, K. M. van Hee and R. Valk, Eds., vol. 5062. Springer,.

[12] R. Milner (2008): *Bigraphs and their algebra*. Proc. of the LIX Colloquium on Emerging Trends in Concurrency Theory, Electronic Notes in Theoretical Computer Science Elsevier, V. 209, 5-19.

[13] R. Milner (2009): *The Space and Motion of Communicating Agents*. Cambridge University Press.

[14] Michele Sevegnani Muffy Calder (2014): *Modelling IEEE 802.11 CSMA-CA RTS-CTS with stochastic bigraphs with sharing*. Formal Aspects of Computing, Volume 26, Issue 3, pp 537-561, Springer.

[15] Maryam Kamali Peter Hoefner (2013): *Quantitative Analysis of AODV and its Variants on Dynamic Topologies using Statistical Model Checking*. 11th International Conference on Formal Modeling and Analysis of Timed Systems (Formats '13), pp. 15, Buenos Aires, Argentina.

[16] Sarah Edenhofer Peter Hoefner (2012): *Towards a Rigorous Analysis of AODVv2 (DYMO)*. 2nd International Workshop on Rigorous Protocol Engineering (WRiPE 2012), pp. 1-6, Austin, Texas.

[17] O.H. Jensen R. Milner (2004): *Bigraphs and mobiles processes*. Technical Report 580, University of Cambridge.

[18] Faiza Belala Rachida Boucebsi, Lakhdar Derdouri (2014): *Affectation des Canaux dans Routage Multi-Chemins: Cas des Reseaux Mesh*. 3rd edition of the Student Day ESI'14 (Jeesi 14, Algiers, Algria).

[19] Lakhdar Derdouri Rachida Boucebsi, Faiza Belala (2014): *Modeling Channel Allocation via BRS: Case of WMNs*. International Conference on Advanced Aspects of Software Engineering (ICAASE14).

[20] Michele Sevegnani (2012): *Bigraph Evaluator and Rewriting*. Available at http://www.dcs.gla.ac.uk/ michele/bigrapher.html.

[21] Michele Sevegnani (2012): *Bigraphs with sharing and applications in wireless networks*. PhD thesis, University of Glasgow.

[22] L. Birkedal T.C. Damgaard (2006): *Axiomatizing binding bigraphs*. Nordic Journal of Computing, 13(12):5877.

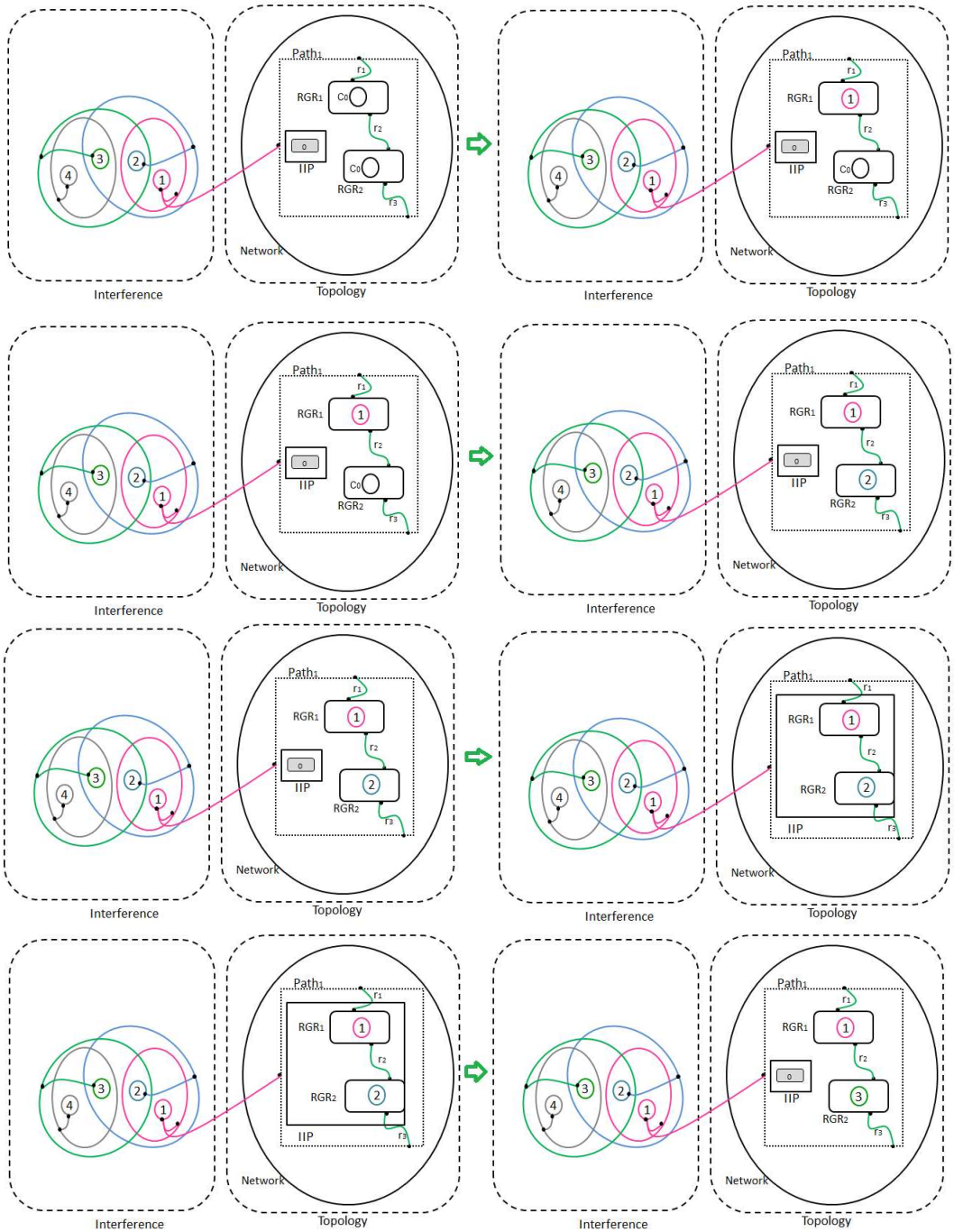


Figure 11: Example of Meta-rules instantiation

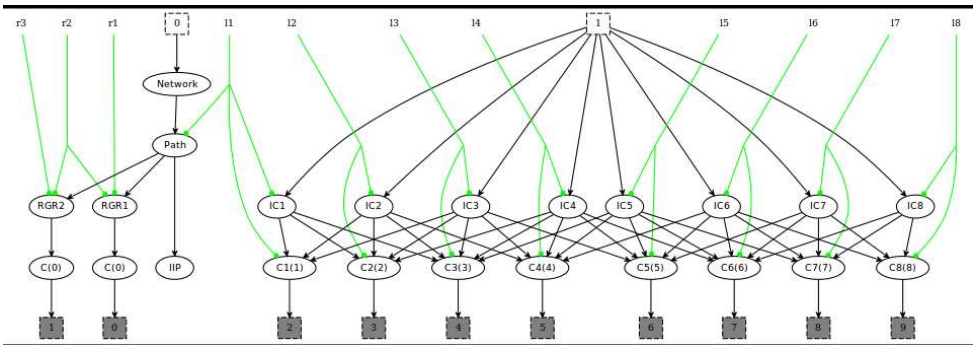
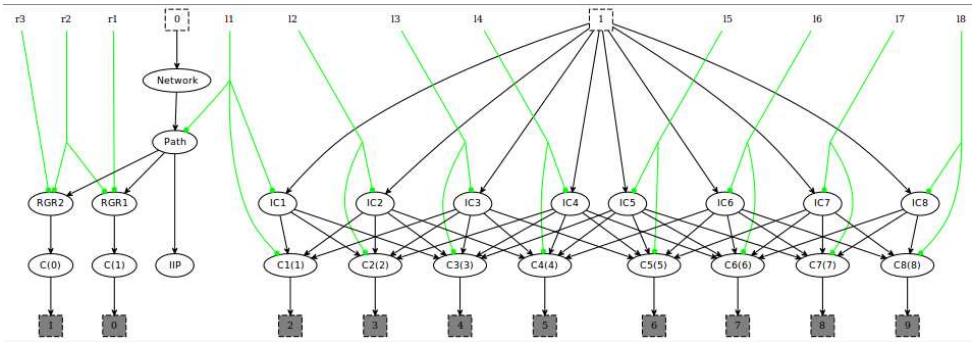
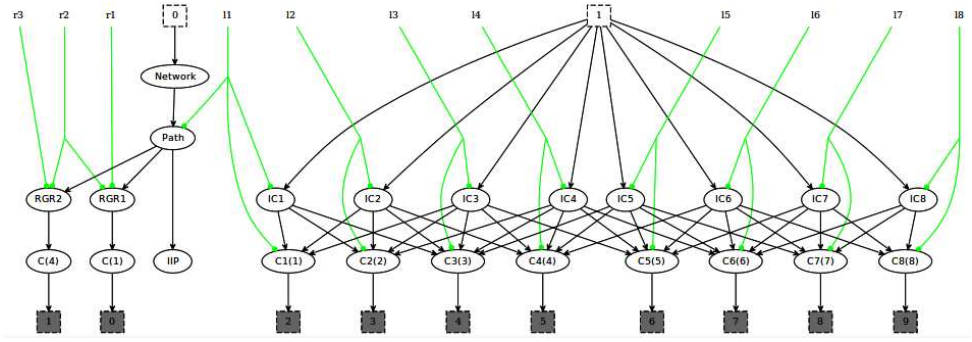
states	Simulation
1	
1	
1	

Table 2: Simulation results

Structured Operational Semantics for Graph Rewriting*

Andrei Dorman
Dip. di Filosofia, Università Roma Tre
LIPN – UMR 7030, Université Paris 13
andrei.dorman@lipn.univ-paris13.fr

Tobias Heindel
LIPN – UMR 7030, Université Paris 13
tobias.heindel@lipn.univ-paris13.fr

Process calculi and graph transformation systems provide models of reactive systems with labelled transition semantics. While the semantics for process calculi is compositional, this is not the case for graph transformation systems, in general. Hence, the goal of this article is to obtain a compositional semantics for graph transformation system in analogy to the structural operational semantics (SOS) for Milner's Calculus of Communicating Systems (CCS).

The paper introduces an SOS style axiomatization of the standard labelled transition semantics for graph transformation systems. The first result is its equivalence with the so-called Borrowed Context technique. Unfortunately, the axiomatization is not compositional in the expected manner as no rule captures "internal" communication of sub-systems. The main result states that such a rule is derivable if the given graph transformation system enjoys a certain property, which we call "complementarity of actions". Archetypal examples of such systems are interaction nets. We also discuss problems that arise if "complementarity of actions" is violated.

Key words: process calculi, graph transformation, structural operational semantics, compositional methods

1 Introduction

Process calculi remain one of the central tools for the description of interactive systems. The archetypal example of process calculi are Milner's π-calculus and the even more basic calculus of communication systems (CCS). The semantics of these calculi is given by labelled transition systems (LTS), which in fact can be given as a structural operational semantics (SOS). An advantage of SOS is their potential for combination with compositional methods for the verification of systems (see e.g. [17]).

Fruitful inspiration for the development of LTS semantics for other "non-standard" process calculi originates from the area of graph transformation where techniques for the derivation of LTS semantics from "reaction rules" have been developed [16, 7]. The strongest point of these techniques is the context independence of the resulting behavioral equivalences, which are in fact congruences. Moreover, these techniques have lead to original LTS-semantics for the ambient calculus [15, 3], which are also given as SOS systems. Already in the special case of ambients, the SOS-style presentation goes beyond the standard techniques of label derivation in [16, 7]. An open research challenge is the development of a general technique for the canonical derivation of SOS-style LTS-semantics. The problem is the "monolithic" character of the standard LTS for graph transformation systems.

In the present paper, we set out to develop a partial solution to the problem for what we shall call CCS-*like* graph transformation systems. The main idea is to develop an analogy to CCS where each action α has a co-action $\overline{\alpha}$ that can synchronize to obtain a silent transition; this is the so-called *communication rule*. In analogy, one can restrict attention to graph transformation systems with rules that allow to assign to each (hyper-)edge a unique *co-edge*. Natural examples of such systems are interaction nets as

*This work was partially supported by grants from Agence Nationale de la Recherche, ref. ANR-08-BLANC-0211-01 (COMPLICE project) and ref. ANR-09-BLAN-0169 (PANDA project).

Bliudze, S., Bruni, R., Carbone, M., Silva, A. (Eds.); ICE 2011
EPTCS 59, 2011, pp. 37–51, doi:10.4204/EPTCS.59.4

introduced by Lafont [11, 1]. In fact, one of the motivations of the paper is to derive SOS semantics for interaction nets.

Structure and contents of the paper We first introduce the very essentials of graph transformation and the so-called Borrowed Context (BC) technique [7] for the special case of (hyper-)graph transformation in Section 2. To make the analogy between CCS and BC as formal as possible, we introduce the system SOSBC in Section 3, which is meant to provide the uninitiated reader with a new perspective on the BC technique. Moreover, the system SOSBC emphasizes the "local" character of graph transformations as every transition can be decomposed into a "basic" action in some context. In particular, we do not have any counterpart to the communication rule of CCS, which shall be addressed in Section 4. We illustrate why it is not evident when and how two labeled transitions of two states that share their interface can be combined into a single synchronized action. However, we will be able to describe sufficient conditions on (hyper-)graph transformation systems that allow to derive the counterpart of the communication rule of CCS in the system SOSBC. Systems of this kind have a natural notion of "complementarity of actions" in the LTS.

2 Preliminaries

We first recall the standard definition of (hyper-)graphs and a formalism of transformation of hyper-graphs (following the double pushout approach). We also present the labelled transition semantics for hyper-graph transformation systems that has been proposed in [7]. In the present paper, the more general case of categories of graph-like structures is not of central importance. However, some of the proofs will use basic results of category theory.

Definition 2.1 (Hypergraphs and hypergraph morphisms)**.** Let Λ be a set of *labels* with associated *arity* function $\mathrm{ar}\colon \Lambda \to \mathbb{N}$. A *($\Lambda$-labelled) hyper-graph* is a tuple $G = (E, V, \ell, \mathrm{cnct})$ where E is a set of *(hyper-)edges*, V is a set of *vertices* or *nodes*, $\ell\colon E \to \Lambda$ is the *labelling function*, and cnct is the *connection* function, which assigns to each edge $e \in E$ a string (e.g. a finite sequence) of *incident vertices* $\mathrm{cnct}(e) = v_1 \cdots v_n$ of length $\mathrm{ar}(\ell(e)) = n$ (where $\{v_1, \ldots, v_n\} \subseteq V$). Let $v \in V$ be a node; its *degree*, written $\deg(v)$ is the number of edges of which it is an incident node, i.e. $\deg(v) = |\{e \in E \mid v \text{ incident to } e\}|$ (where for any finite set M, the number of elements of M is $|M|$). We also write $v \in G$ and $e \in G$ if $v \in V$ and $e \in E$.

Let $G_i = (E_i, V_i, \ell_i, \mathrm{cnct}_i)$ ($i \in \{1,2\}$) be hyper-graphs; a *hyper-graph morphism* from G_1 to G_2, written $f\colon G_1 \to G_2$ is a pair of functions $f = (f_E\colon E_1 \to E_2, f_V\colon V_1 \to V_2)$ such that $\ell_2 \circ f_E = \ell_1$ and for each edge $e_1 \in E_1$ with attached nodes $\mathrm{cnct}(e) = v_1 \cdots v_n$ we have $\mathrm{cnct}_2(f_E(e)) = f_V(v_1) \cdots f_V(v_n)$. A hyper-graph morphism $f = (f_E, f_V)\colon G_1 \to G_2$ is *injective (bijective)* if both f_E and f_V are injective (bijective); it is an inclusion if both $f_E(e) = e$ and $f_V(v) = v$ hold for all $e \in E_1$ and $v \in V_1$. We write $G_1 \to G_2$ or $G_2 \leftarrow G_1$ if there is an inclusion from G_1 to G_2, in which case G_1 is a *sub-graph* of G_2.

To define double pushout graph transformation and the Borrowed Context technique [7], we will need the following constructions of hyper-graphs, which roughly amount to intersection and union of hyper-graphs.

Definition 2.2 (Pullbacks & pushouts of monos)**.** Let $G_i = (E_i, V_i, \ell_i, \mathrm{cnct}_i)$ ($i \in \{0,1,2,3\}$) be hyper-graphs and let $G_1 \to G_3 \leftarrow G_2$ be inclusions. The *intersection of G_1 and G_2* is the hyper-graph $G' = (E_1 \cap E_2, V_1 \cap V_2, \ell', \mathrm{cnct}')$ where $\ell'(e) = \ell_1(e)$ and $\mathrm{cnct}'(e) = \mathrm{cnct}_2(e)$ for all $e \in E_1 \cap E_2$. The *pullback of $G_1 \to G_3 \leftarrow G_2$* is the pair of inclusions $G_1 \leftarrow G' \to G_2$ and the resulting square is a pullback square (see Figure 1).

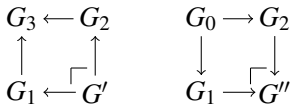


Figure 1: Pullback and pushout square

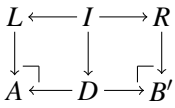
Let $G_1 \leftarrow G_0 \rightarrow G_2$ be inclusions; they are *non-overlapping* if both $E_1 \cap E_2 \subseteq E_0$ and $V_1 \cap V_2 \subseteq V_0$ hold. The *pushout of non-overlapping inclusions* $G_1 \leftarrow G_0 \rightarrow G_2$ is the pair of inclusions $G_1 \rightarrow G'' \leftarrow G_2$ where $G'' = (E_1 \cup E_2, V_1 \cup V_2, \ell'', \mathrm{cnct}'')$ is the hyper-graph that satisfies

$$\ell''(e) = \begin{cases} \ell_1(e) & \text{if } e \in E_1 \\ \ell_2(e) & \text{otherwise} \end{cases} \quad \text{and} \quad \mathrm{cnct}''(e) = \begin{cases} \mathrm{cnct}_1(e) & \text{if } e \in E_1 \\ \mathrm{cnct}_2(e) & \text{otherwise} \end{cases}$$

for all $e \in E_1 \cup E_2$.

Finally, we are ready to introduce graph transformation systems and their labelled transition semantics.

Definition 2.3 (Rules and graph transformation systems)**.** A *rule (scheme)* is a pair of non-overlapping inclusions of hyper-graphs $\rho = (L \leftarrow I \rightarrow R)$. Let A, B be hyper-graphs such that $A \leftarrow L$ and moreover $A \leftarrow I \rightarrow R$ is non-overlapping. Now, ρ *transforms A to B* if there exists a diagram as shown on the right such that the two squares are pushouts and there is an isomorphism $\iota : B' \rightarrow B$. A *graph transformation system* (GTS) is pair $\mathscr{S} = (\Lambda, \mathscr{R})$ where Λ is a set of labels and $\mathscr{R}$ is a set of rules.

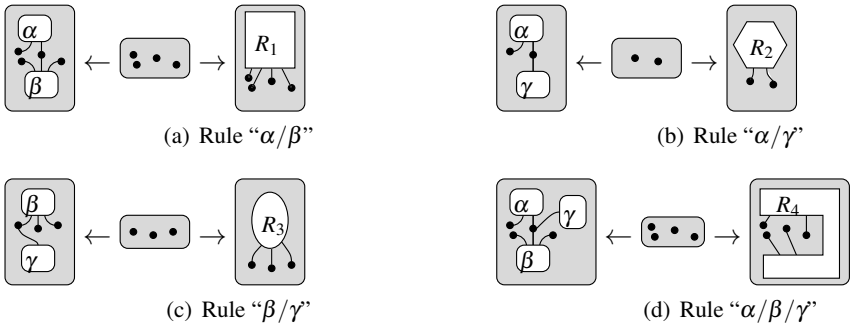


A graph transformation rule can be understood as follows. Whenever the left hand side L is (isomorphic to) a sub-graph of some graph A then this sub-graph can be "removed" from A, yielding the graph D. The vacant place in D is then "replaced" by the right hand side R of the rule. The middleman I is the memory of the connections L had with the rest of the graph in order for R to be attached in exactly the same place.

We now present an example that will be used throughout the paper to illustrate the main ideas.

Example 2.1. The system $\mathscr{S}_{ex} = (\Lambda, \mathscr{R})$ will be the following one in the sequel: $\Lambda = \{\alpha, \beta, \gamma, \dots\}$ such that $\mathtt{ar}(\alpha) = 2$, $\mathtt{ar}(\beta) = 3$ and $\mathtt{ar}(\gamma) = 1$; moreover $\mathscr{R}$ is the set of rules given in Figure 2 where the R_i represent different graphs (e.g. edges with labels R_i).

To keep the graphical representations clear, all inclusions in the running example are given implicitly by the spatial arrangement of nodes and edges.



(a) Rule "α/β"

(b) Rule "α/γ"

(c) Rule "β/γ"

(d) Rule "$\alpha/\beta/\gamma$"

Figure 2: Reaction rules of $\mathscr{S}_{ex}$.

Remark 2.1 (Rule instances). Given a rule $L \leftarrow I \rightarrow R$ and a graph A such that $A \leftarrow L$, one can assume w.l.o.g. that $A \leftarrow I \rightarrow R$ is non-overlapping. The reason is that in each case, the rule $L \leftarrow I \rightarrow R$ could be replaced by an *isomorphic* "rule instance" $\rho' = L' \leftarrow I' \rightarrow R'$ (based on the standard notion of rule isomorphism).

In fact the result of each transformation step is unique (up to isomorphism). This is a consequence of the following fact.

Fact 2.4 (Pushout complements). *Let $G_2 \leftarrow G_1 \leftarrow G_0$ be a pair of hyper-graph inclusions where $G_i = (E_i, V_i, \ell_i, \mathrm{cnct}_i)$ ($i \in \{0,1,2\}$) such that for all $v \in V_1 \setminus V_0$ there does not exist any edge $e \in E_2 \setminus E_0$ such that v is incident to e. Then there exists a unique sub-graph $G_2 \leftarrow D$ such that* (1) *is a pushout square.*

$$\begin{array}{ccc} G_1 & \longleftarrow & G_0 \\ \downarrow & \ulcorner & \downarrow \\ G_2 & \longleftarrow & D \end{array} \qquad (1)$$

Definition 2.5 (Pushout Complement). Let $G_2 \leftarrow G_1 \leftarrow G_0$ be a pair of hyper-graph inclusions that satisfy the conditions of Fact 2.4; the unique completion $G_2 \leftarrow D \leftarrow G_0$ in (1) is the *pushout complement* of $G_2 \leftarrow G_1 \leftarrow G_0$.

Definition 2.6 (Labelled transition system). A *labelled transition system* (LTS) is a tuple $(S, Ł, R)$ where S is a set of *states*, Ł is a set of *labels* and $R \subseteq S \times Ł \times S$ is the *transition relation*. We write

$$s \xrightarrow{\alpha} s'$$

if $(s, \alpha, s') \in R$ and say that s can evolve to s' by performing α.

Definition 2.7 (DPOBC). Let $\mathscr{S} = (\Lambda, \mathscr{R})$ be a graph transformation system. Its LTS has all inclusions of hyper-graphs $J \rightarrow G$ as *states* where J is called the *interface*; the labels are all pairs of inclusions $J \rightarrow F \leftarrow K$, and a state $J \rightarrow G$ evolves to another one $K \rightarrow H$ if there is a diagram as shown on the right, which is called a DPOBC-*diagram* or just a BC-*diagram*. In this diagram, the graph D is called the *partial match of L*.

$$\begin{array}{ccccccc} D & \longrightarrow & L & \longleftarrow & I & \longrightarrow & R \\ \downarrow & & \downarrow & \ulcorner & \downarrow & \ulcorner & \downarrow \\ G & \longrightarrow & G_c & \longleftarrow & C & \longrightarrow & H \\ \uparrow & & \uparrow & \llcorner & \uparrow & \ulcorner & \nearrow \\ J & \longrightarrow & F & & \longleftarrow & K & \end{array}$$

For a technical justification of this definition, see [16], but let us give some intuitions on what this diagram expresses. States are inclusions, where the "larger" part models the whole "internal" state of the system while the "smaller" part, the interface, models the part that is directly accessible to the environment and allows for (non-trivial) interaction. As a particular simple example, one could have a Petri net where the set of places (with markings) is the complete state and some of the place are "open" to the environment such that interaction takes place by exchange of tokens.

The addition of agents/resources from the environment might result in "new" reactions, which have not been possible before. The idea of the LTS semantics for graph transformation is to consider (the addition of) "minimal" contexts that allow for "new" reactions as labels. The minimality requirement of an addition $J \rightarrow E$ or $J \rightarrow F$ is captured by the two leftmost squares in the BC diagram above: the addition $J \rightarrow F$ is "just enough" to complete part of the left hand side L of some rule. If the reaction actually takes place, which is captured by the other two squares in the upper row in the BC diagram, some agents might disappear / some resources might be used (depending on the preferred metaphor) and new ones might appear. Finally the pullback square in the BC diagram restricts the changes to obtain the new interface into the result state after reaction. As different rules might result in different deletion effects that are "visible" to the environment, the full label of each such "new" reaction is the "trigger" $J \rightarrow F$ together with the "observable" change $F \leftarrow K$ (with state $K \rightarrow H$ after interaction).

3 Three Layer SOS semantics

We start with a reformulation of the borrowed context technique that breaks the "monolithic" BC-step into axioms (that allow to derive the *basic actions*) and two rules that allow to perform these basic actions within suitable contexts. The axioms corresponds to the CCS-axioms that describe that the process $\alpha.P$ can perform the action α and then behaves as P, written $\alpha.P \xrightarrow{\alpha} P$ where α ranges over the actions $a, \overline{a}$, and τ. In the case of graphs, each rule $L \leftarrow I \rightarrow R$ gives rise to such a set of actions. More precisely, each subgraph D of L can be seen as an "action" with co-action $\widehat{D}^L \rightarrow L$ such that L is the union of D and $\widehat{D}^L$. For example, in the rule α/β, both edges α and β yield (complementary) basic actions.

Formally, in Table 1, we have the family of *Basic Action* axioms. It essentially represents all the possible uses of a transformation rule. In an (encoding of) CCS, the left hand side would be a pair of unary edges a and $\overline{a}$, which both disappear during reaction. Now, if only a is present "within" the system, it needs $\overline{a}$ to perform a reaction; thus, the part a of the left hand side induces the (inter-)action that consists in "borrowing" $\overline{a}$ and deleting both edges (and similarly for $\overline{a}$). In general, e.g. in the rule $\alpha/\beta/\gamma$ there might be more than two edges that are involved in a reaction and thus we have a whole family of actions. More precisely, each portion of a left hand side induces the action that consists in borrowing the missing part to perform the reaction (thus obtaining the coplete left hand side), followed by applying the changes that are described by the right part of the rule.

Next, we shall give counterparts for two CCS-rules that describe that an action can be performed in parallel to another process and under a restriction. More precisely, whenever we have the transition $P \xrightarrow{\alpha} P'$ and another process Q, then there is also a transition $P \parallel Q \xrightarrow{\alpha} P' \parallel Q$; similarly, we also have $(\nu b)P \xrightarrow{\alpha} (\nu b)P'$ whenever $\alpha \notin \{\overline{b}, b\}$. More abstractly, actions are preserved by certain contexts. The notion of context in the case of graph transformation, which will be the counterpart of process contexts such as $P \parallel [\cdot]$ and $(\nu b)[\cdot]$, is as follows.

Definition 3.1 (Context)**.** A *context* is a pair of inclusions $C = J \rightarrow E \leftarrow J'$. Let $J \rightarrow G$ be a state (such that $E \leftarrow J \rightarrow G$ is non-overlapping); the *combination* of $J \rightarrow G$ with the context C, written $C[J \rightarrow G]$, is the inclusion of J' into the pushout of $E \leftarrow J \rightarrow G$ as illustrated in the following display.

$$\text{state: } \begin{array}{c} G \\ \uparrow \\ J \end{array} \qquad \text{context: } J \longrightarrow E \longleftarrow J' \qquad \text{construction: } \begin{array}{ccc} G & \longrightarrow & \overline{G} \\ \uparrow & & \uparrow \\ J & \longrightarrow & E \longleftarrow J' \end{array} \qquad \text{combination: } \begin{array}{c} \overline{G} \\ \uparrow \\ J' \end{array}$$

The left inclusion of the context, i.e. $J \rightarrow E$, can also be seen as a state with the same interface. The pushout then gives the result of "gluing" E to the original G at the interface J; the second inclusion $J' \rightarrow E$ models a new interface, which possibly contains part of J and additional "new" entities in E.

With this general notion of context at hand, we shall next address the counterpart of name restriction, which we call *interface narrowing*, the second rule family in Table 1. In CCS, the restriction (νa) preserves only those actions that do not involve a. The counterpart of the context $(\nu a)[\cdot]$ is a context of the form $J \rightarrow J \leftarrow J'$. In certain cases, one can "narrow" a label while "maintaining" the "proper" action as made formal in the following definition.

Definition 3.2 (Narrowing)**.** A *narrowing context* is a context of the form $C = J \rightarrow J \leftarrow J'$. Let $J \rightarrow F \leftarrow K$ be a label such that the pushout complement of $F \leftarrow J \leftarrow J'$ exists; then the *C-narrowing* of the label, written $C[J \rightarrow F \leftarrow K]$ is the lower row in the following display

$$C[J \rightarrow F \leftarrow K] := \begin{array}{ccccc} J & \longrightarrow & F & \longleftarrow & K \\ \uparrow & & \uparrow & & \uparrow \\ J' & \longrightarrow & F' & \longleftarrow & K' \end{array} \qquad \text{where } C = J \rightarrow J \leftarrow J'$$

where the left square is a pushout and the right one a pullback. Whenever we write $C[J \to F \leftarrow K]$, we assume that the relevant pushout complement exists.

If we think of the interface as the set of free names of a process, then restricting a name means removal from the interface. Thus, J' is the set of the remaining free names. If the pushout complement F' exists, it represents F with the restricted names erased. Finally, since a pullback here can be seen as an intersection, K' is K without the restricted names. So we finally obtain the "same" label where "irrelevant" names are not mentioned. It is of course not always possible to narrow the interface. For instance, one cannot restrict the names that are involved in labelled transitions of CCS-like process calculi. This impossibility is captured by the non-existence of the pushout complement.

With the notion of narrowing, we can finally define the interface narrowing rule in Table 1.

The final rule in Table 1 captures the counterpart of performing an action in parallel composition with another process P. In the case of graph transformation, this case is non-trivial since even the pure addition of context potentially interferes with the action of some state $J \to G$. For example, if an interaction involves the deletion of an (isolated) node, the addition of an edge to this node inhibits the reaction. However, for each transition there is a natural notion of non-inhibiting context; moreover, to stay close to the intuition that parallel composition with a process P only adds new resources and to avoid overlap with the narrowing rule, we restrict to monotone contexts.

Definition 3.3 (Compatible contexts)**.** Let $C = J \to E \leftarrow \overline{J}$ be a context; it is *monotone* if $J \to \overline{J}$. Let $J \to F \leftarrow K$ be a label; now C is *non-inhibiting w.r.t.* $J \to F \leftarrow K$ if it is possible to construct the diagram (2) where both squares are pushouts. Finally, a context $J \to E \leftarrow \overline{J}$ is *compatible* with the label $J \to F \leftarrow K$ if it is non-inhibiting w.r.t. it and monotone.

$$\begin{array}{ccccc} J & \longrightarrow & F & \longleftarrow & K \\ \downarrow & & \downarrow & & \downarrow \\ E & \longrightarrow & E_1 & \longleftarrow & E' \end{array} \qquad (2)$$

In a label $J \to F \leftarrow K$, the left inclusion represents the addition of new entities that "trigger" a certain reaction. A compatible context is simply a context that is able to provide at least F, usually more than F, while not attaching new edges to nodes that disappear during reaction.

The last rule in the SOSBC-system of Table 1 is the embedding of a whole transition into a monotone context. To define this properly, we introduce a partial operation for the "combination" of co-spans (which happens to be a particular type of relative pushout of co-spans); this generalizes the narrowing construction.

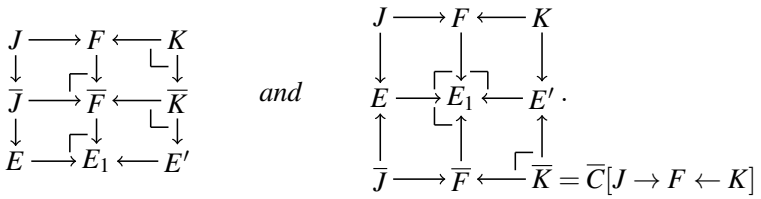
Definition 3.4 (Cospan combination)**.** Let $C = (J \to F \leftarrow K)$ and $\overline{C} = (J \to E \leftarrow \overline{J})$ be two cospans. They are *combinable* if there exists a diagram of the following form.

$$\begin{array}{ccccc} J & \longrightarrow & F & \longleftarrow & K \\ \downarrow & & \downarrow & & \downarrow \\ E & \longrightarrow & E_1 & \longleftarrow & E' \\ \uparrow & & \uparrow & & \uparrow \\ \overline{J} & \longrightarrow & \overline{F} & \longleftarrow & \overline{K} \end{array} =: \overline{C}[J \to F \leftarrow K]$$

The label $\overline{J} \to \overline{F} \leftarrow \overline{K}$ is the *combination of C with $\overline{C}$*, and is denoted by $\overline{C}[J \to F \leftarrow K]$.

In fact, it is easy to show that compatible contexts are combinable with their label.

Lemma 3.5. *Given a reduction label $J \to F \leftarrow K$ and a compatible context $J \to E \leftarrow \overline{J}$ for it, we can split the diagram* 2 *in order to get*



With this lemma we can finally define the rule that corresponds to "parallel composition" of an action with another "process". Now the SOSBC-system does not only give an analogy to the standard SOS-semantics for CCS, we shall also see that the labels that are derived by the standard BC technique are exactly those labels that can be obtained from the basic actions by compatible contextualization and interface narrowing. In technical terms, the SOSBC-system of Table 1 is sound and complete.

- **Basic Actions**

$$\frac{}{(D \to D) \xrightarrow{D \to L \leftarrow I} (I \to R)} \qquad \text{where } (L \leftarrow I \to R) \in \mathscr{S} \text{ and } D \to L$$

- **Interface Narrowing**

$$\frac{(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)}{(J' \to G) \xrightarrow{J' \to F' \leftarrow K'} (K' \to H)} \qquad \text{where } C = J \to J \leftarrow J' \text{ and } J' \to F' \leftarrow K' = C[J \to F \leftarrow K]$$

- **Compatible Contextualization**

$$\frac{(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)}{C[J \to G] \xrightarrow{C[J \to F \leftarrow K]} \overline{C}[K \to H]} \qquad \text{where } C = J \to E \leftarrow \overline{J} \text{ compatible with } J \to F \leftarrow K \text{ and } \overline{C} = (J \to F \leftarrow K)[C]$$

Table 1: Axioms and rules of the SOSBC-system.

Theorem 3.6 (Soundness and completeness)**.** *Let $\mathscr{S}$ be a graph transformation system. Then there is a* BC*-transition*

$$(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$$

if and only if it is derivable in the SOSBC*-system.*

The main role of this theorem is not its technical "backbone", which is similar to many other theorems on the Borrowed Context technique. The main insight to be gained is the absence of any "real" communication between sub-systems; roughly, every reaction of a state can be "localized" and then derived from a basic action (followed by contextualization and narrowing). In particular, we do not have any counterpart to the communication-rule in CCS, which has complementary actions $P \xrightarrow{a} P'$ and $Q \xrightarrow{\overline{a}} Q'$ as premises and concludes the possibility of communication of the processes P and Q to perform the silent "internal" transition $P \parallel Q \xrightarrow{\tau} P' \parallel Q'$. The main goal is to provide an analysis of possible issues with a counterpart of this rule.

4 The composition rule for CCS-like systems

Process calculi, such as CCS and the π-calculus, have a so-called *communication rule* that allows to synchronize sub-processes to perform silent actions. The involved process terms have complementary actions that allow to interact by a "hand-shake". However, it is an open question how such a communication rule can be obtained for general graph transformations systems via the Borrowed Context technique. Roughly, the label of a transition does not contain information about which reaction rule was used to derive it; in fact, the same label might be derived using different rules. Intuitively, we do not know how to identify the two hands that have met to shake hands.

To elaborate on this using the metaphor of handshakes, assume that we have an agent that needs a hand to perform a handshake or to deliver an object. If we observe this agent reaching out for another hand, we cannot conclude from it which of the two possible actions will follow. In general, even after the action is performed, it still is not possible to know the decision of the agent – without extra information, which might however not be observable. However, with suitable assumptions about the "allowed actions", all necessary information might be available.

First, we recall from [2] that DPOBC-diagrams (as defined in Definition 2.7) can be composed under certain circumstances.

Fact 4.1. *Let*

$$(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H) \quad \text{and} \quad (J' \to G') \xrightarrow{J' \to F' \leftarrow K'} (K' \to H')$$

be two transitions obtained from two DPOBC*-diagrams with the same rule $\rho = L \leftarrow I \to R$. Then, it is possible to build a* DPOBC*-diagram with the same rule for the composition of $J \to G$ and $J' \to G'$ along some common interface $J \leftarrow J^L_D \to J'$.*

Take the following example as illustration of this fact.

Example 4.1 (Composition of transitions)**.** Let $J \to G$ be a state of $\mathscr{S}_{ex}$ that contains an edge α with its second connection in the interface as shown in Figure 3(a). Further, let $J' \to G'$ be a state that contains an edge β with its second connection in the interface as shown in Figure 3(b). Both graphs can trigger a reaction from rule $\alpha/\beta/\gamma$. Such a composition is shown in Figure 3(c).

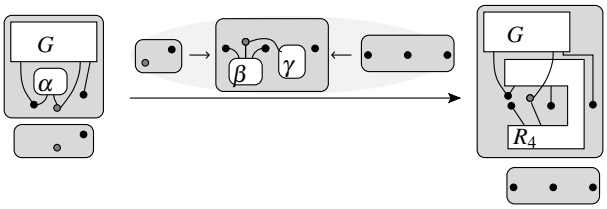
Hence, we see that is in general possible to combine transitions to obtain new transitions. However, we emphasize at this point, that derivability of a counterpart of the communication rule of CCS is not the same question as the composition of pairs of transitions that come equipped with *complete* BC-diagrams. To clarify the problem, consider the following example where we cannot infer the used rule from the transition label.

Example 4.2. Let G be a graph composed of two edges α and β and consider a transition label where an edge γ is "added". Then it is justified by both rules α/γ and β/γ (see Figure 4).

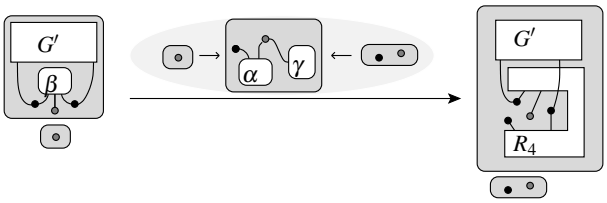
We shall avoid this problem by restricting to suitable classes of graph transformation systems. Moreover, for simplicities sake, we shall focus on the derivation of "silent" transitions in the spirit of the communication rule of CCS.

Definition 4.2 (Silent label)**.** A label $J \to F \leftarrow K$ is *silent* or τ if $J = F = K$; a *silent transition* is a transition with a silent label.

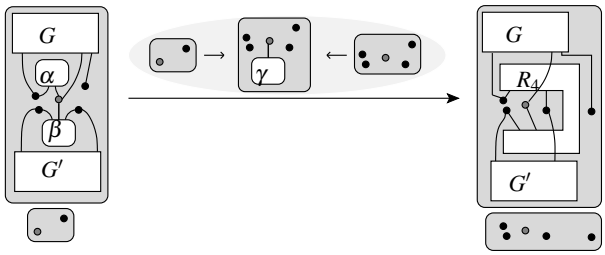
Intuitively, a silent transition is one that does not induce any "material" change that is visible to an external observer that only has access to the interface of the states. Hence, in particular, a silent transition does not involve additions of the environment during the transition. Moreover, the interface remains



(a) A first transition



(b) A second transition



(c) The composition of the transitions

Figure 3: An example of composition.

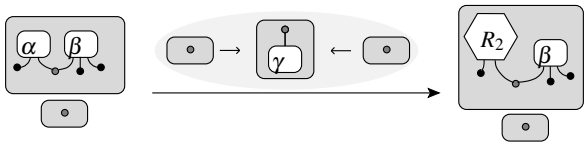
unchanged. This latter requirement does not have any counterpart in process calculi, as the interface is given implicitly by the set of all free names. (In graphical encodings of process terms [3] it is possible to have free names in the interface even though there is no corresponding input or output prefix in the term.)

Now, with the focus on silent transitions, for a given rule $L \leftarrow I \rightarrow R$ we can illustrate the idea of complementary actions as follows. If a graph G contains a subgraph D of L and moreover a graph G' has the complementary subgraph of D in L in it, then G and G' can be combined to obtain a big graph $\overline{G}$ – the "parallel composition" of G and G' – that has the whole left hand side L as a subgraph and thus $\overline{G}$ can perform the reaction. A natural example for this are Lafont's interaction nets where the left hand side consist exactly of two hyper-edges, which in this case are called cells. The intuitive idea of complementary (basic) actions is captured by the notion of *active pairs*.

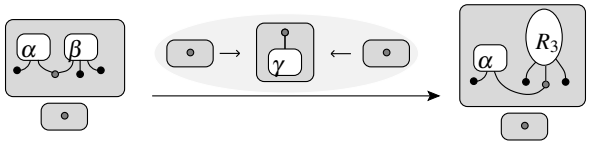
Definition 4.3 (Active pairs)**.** For any inclusion $D \rightarrow L$, where $D \neq L$ and for all nodes v of D, $\deg(v) > 0$, let the following square be its initial pushout

$$\begin{array}{ccc} J_D^L & \longrightarrow & \widehat{D}^L \\ \downarrow & & \downarrow \\ D & \longrightarrow & L \end{array},$$

i.e. $\widehat{D}^L$ is the smallest subgraph of L that allows for completion to a pushout. We call $\widehat{D}^L$ the *complement of D in L* and J_D^L the *minimal interface of D in L* and we write $\{D, D'\} \equiv L$ if $D' = \widehat{D}^L$. The set of *active*



(a) A transition from rule α/γ



(b) A transition from rule β/γ

Figure 4: Same transition label for different rules.

pairs is

$$\mathbb{D} = \left\{ \{D, \widehat{D}^L\} \mid L \leftarrow I \rightarrow R \in \mathscr{R}, D \rightarrow L,\ D \neq L,\ \forall v \in D.\, \deg(v) > 0 \right\}.$$

Abusing notation, we also denote by $\mathbb{D}$ the union of $\mathbb{D}$.

It is easy to verify that the complement of $\widehat{D}^L$ in L is D itself and that its minimal interface is also J^L_D. It is the set of "acceptable" partial matches in the sense that they do not yield a τ-reaction on their own. Indeed, if D is equal to L, then the resulting transition of this partial match is a τ-transition. And if it is just composed of vertices, its complement is L and thus not acceptable.

Example 4.3 (Active pairs)**.** In our running example, the set $\mathbb{D}$ of our example is in obvious bijection to

$$\big\{\{\alpha,\beta\},\{\alpha,\gamma\},\{\beta,\gamma\},\{\alpha,\beta+\gamma\},\{\alpha+\beta,\gamma\},\{\alpha+\gamma,\beta\}\big\}.$$

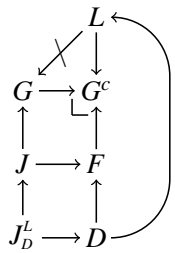
The minimal interface of any pair is a single vertex.

This completes the introduction of preliminary concepts to tackle the issues that have to be resolved to obtain "proper" compositionality of transitions.

4.1 Towards a partial solution

Let us address the problem of identifying the rule that is "responsible" for a given interaction. We start by considering the left inclusions of labels, which intuitively describe possible borrowing actions from the environment. Relative to this, we define the *admissible rules* as those rules that can be used to let states evolve while borrowing the specified "extra material" from the environment.

Definition 4.4 (Admissible rule)**.** Let $J \rightarrow G$ be a state and let $J \rightarrow F$ be an inclusion (which represents a possible contribution of the context). A rule ρ is *admissible (for $J \rightarrow F$)* if $L \not\rightarrow G$ and it is possible to find $D \in \mathbb{D}$ and L the left-hand side of ρ, such that the following diagram commutes



where $J_D^L \to D$ is the minimal interface of D in L. We call D the *rule addition*.

This just means that G *can* evolve using the rule ρ if D is added at the proper location.

Proposition 4.5 (Precompositionality)**.** *Let $J \to G \xrightarrow{J\to F\leftarrow K} K \to H$ and $J' \to G' \xrightarrow{J'\to F'\leftarrow K'} K' \to H'$ be two transitions such that a single rule ρ is admissible for both, and let D and D' be their respective rule additions. If $\{D,D'\} \in \mathbb{D}$, it is possible to compose G and G' into a graph $\overline{G}$ in a way to be able to derive a τ-transition using rule ρ.*

Proof. We first show that in such a case, $D' \to G$ and the pushout of $G \leftarrow D' \to L$ is exactly G^c. Similarly, $D \to G'$ and the pushout of $G' \leftarrow D \to L$ is exactly G'^c. Then, it is easy to see that it is possible to build the DPOBC-diagram $\mathtt{D}_1$ using rule ρ on G (respectively G') yelding the transition $(J \to G) \xrightarrow{J\to F\leftarrow K_1} (K_1 \to H_1)$ for some K_1, H_1 (respectively the DPOBC-diagram $\mathtt{D}_2$ yelding the transition $(J \to G) \xrightarrow{J\to F\leftarrow K_2} (K_2 \to H_2)$ for some K_2, H_2), and then compose $\mathtt{D}_1$ and $\mathtt{D}_2$.

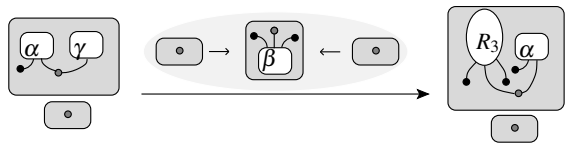
This follows from $\{D,D'\} \in \mathbb{D}$ and $\overline{G} \equiv \overline{G^c}$. Indeed, $\overline{E} = L$ so the top left morphism of the composed DPOBC-diagram is an isomorphism and so are the ones under it, using basic pushout properties. □

This first result motivates the following definition.

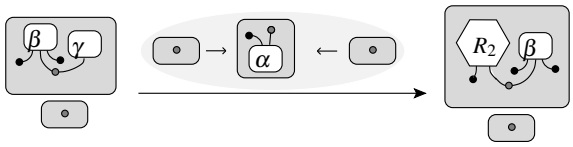
Definition 4.6 (τ-compatible)**.** In the situation of Proposition 4.5, we say the two transitions are τ-*compatible*.

Remark 4.1. In general, in Proposition 4.5, the result of the τ-transition cannot be constructed from H and H'; thus we do not yet speak of compositionality.

Example 4.4. Let G be a graph composed of two edges α and γ and G' of two edges β and γ (see Figure 5). Then the rule α/β is admissible for both transitions and moreover they are τ-compatible. The rule α/β yields the respective rule additions. "Glueing" G and G' by their interface results in a graph with edges α, β and two γs; the latter graph can perform a τ-reaction from rule α/β, which however does not give the desired result since the target state is not the "expected composition" of H and H'. In other words, although we have been able to construct a τ-transition, it is not the composition of the original transitions.



(a) A transition from rule β/γ



(b) A transition from rule α/γ

Figure 5: τ-compatible, but not composable: different rules.

We can see from the examples here that the difficulty of defining a composition of transitions comes mainly from three facts. The first is that a partial match can have several subgraphs triggering a reaction. This is delt with by the construction of the set of active pairs. The second one is the possibility to connect multiple edges together, not knowing which one exactly is consumed in the reaction. Finally, a given edge can have multiples ways of triggering a reaction.

4.2 Sufficient conditions

We now give two frameworks in which neither of the two last problems do occur. Avoiding each of them separately is enough to define compositionality properly. Both cases are inspired by the study of interaction net systems [12, 6, 14], which can be represented in the obvious manner as graph transformation systems. In these systems, the DPOBC-diagram built from an admissible rule of a transition is necessarily the one that has to be used to derive the transition. In one case, it works for essentially the same reasons as in CCS: every active element can only interact with a unique other element, such as a vs. $\overline{a}$, b vs. $\overline{b}$. In the other one, the label itself is not enough, but since we also know where it "connects" to the graph, it is possible to "find" the partner that was involved in the transition.

We introduce interaction graph systems, which are caracterized among other rewriting systems by the form of the left-hand sides of the reaction rules, composed of exactly two hyperedges connected by a single node. We fix a labeling alphabet Λ.

Definition 4.7. An *activated pair* is a hypergraph L on Λ composed of two hyperedges e and f and a node v such that v appears exactly once in $\mathrm{cnct}(e)$ and once in $\mathrm{cnct}(f)$. If v is the i-th incident vertex of e labelled α and the j-th incident vertex of f labelled β, we denote the activated pair by $e_i \bowtie f_j$ and label it by $\alpha_i \bowtie \beta_j$.

An *interaction graph system* $(\Lambda, \mathscr{R})$ is given by a set of reaction rules $\mathscr{R}$ over hypergraphs on Λ where all left-hand side of rules are activated pairs, and nodes are never deleted, i.e. for any rule $\rho = L \leftarrow I \rightarrow R$,

- L is an activated pair;
- for any node v, $v \in L \Rightarrow v \in I$.

Note that for any interaction graph system, the set $\mathbb{D}$ is composed of pairs $\{D, D'\}$ where each of them is composed of an edge and its connected vertices. Also the minimal interface of any active pair $\{D, D'\}$ is a single node. It is also the case that it is enough for interfaces to be composed of vertices only.

Example 4.5. SIMPLY WIRED HYPERGRAPHS Lafont interaction nets are historically the first interaction nets. They appear as an abstraction of linear logic proof-nets [12]. Originally, Lafont nets have several particular features, but the one we are interested in is the condition on connectivity.

Definition 4.8. Let $N = (E, V, \ell, \mathrm{cnct})$ be a hypergraph on Λ.

The graph N is *simply wired* if $\forall v \in V$, $\deg(v) \leq 2$. When $\deg(v) = 1$, we say that v is *free*.

In other words, vertices are only incident to at most two edges of a graph. Note that in this special case no issues arise if we restrict to the sub-category of simply wired hypergraphs. For this, we argue that the purpose of the interface is the possible addition of extra context; thus, in simply wired hypergraphs, it is meaningless for a vertex that is already connected to two edges to be in the interface.

Definition 4.9 (Lafont interaction graph system)**.** A *Lafont interaction graph* is a simply connected graph such that its interface consists of free vertices only. A *Lafont system* $\mathbb{L} = (\Lambda, \mathscr{R})$ is given by reaction rules over Lafont interaction graphs; it is *partitioned* if two left-hand sides only overlap trivially, i.e. for two rules $\rho_j = L_j \leftarrow I_j \rightarrow R_j \in \mathscr{R}$ $(j = 1, 2)$, either $L_1 = L_2$ or $L_1 \cap L_2$ is the empty graph (without any nodes and any hyperedges).

Lemma 4.10. *Let $\mathbb{L}$ be a partitioned Lafont system, let $J \rightarrow G$ be a state, let $(J \rightarrow G) \xrightarrow{J \rightarrow F \leftarrow K} (K \rightarrow H)$ be a non-τ transition. Then there is exactly one admissible rule for this transition.*

Example 4.6. HYPERGRAPHS WITH UNIQUE PARTNERS By generalizing Lafont interaction nets, we obtain so called *multiwired* interaction nets. But then we lose the unicity of the rule for a given transition label. It can be recovered by another condition.

Definition 4.11 (Unique partners)**.** Let $\mathbb{I} = (\Lambda, \mathscr{R})$ be an interaction graph system. We say it is *with unique partners* if for any $\alpha \in \Lambda$ and for all $i \leq \mathtt{ar}(\alpha)$, there exists a unique $\beta \in \Lambda$ and a unique $j \leq \mathtt{ar}(\beta)$ such that $\alpha_i \bowtie \beta_j$ is the label of a left-hand side of a rule in $\mathscr{R}$.

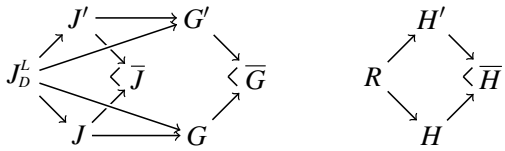
Lemma 4.12. *Let $J \to G$ a state of $\mathbb{I}$ and $(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$ a non-τ reaction label. Then there is exactly one admissible rule ρ for this transition.*

Finally, we conclude our investigation with the following positive result.

Theorem 4.13 (Compositionality)**.** *Let $(\Lambda, \mathscr{R})$ be a Lafont interaction graph system, or an interaction graph system with unique partners. Let $\mathbb{D}$ be its set of active pairs.*

Let $t_1 = (J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$ and $t_2 = (J' \to G') \xrightarrow{J' \to F' \leftarrow K'} (K' \to H')$ be two non-τ transitions and D and D' their respective rule additions.

If $\{D, D'\} \equiv L \in \mathbb{D}$, let $\overline{G}$ and $\overline{H}$ are described by the following diagrams



where $J_D^L \to J$ and $J_D^L \to J'$ are the inclusions from the admissibility of ρ for states $J \to G$ and $J' \to G'$ (Definition 4.4).

Then

$$(\overline{J} \to \overline{G}) \xrightarrow{\overline{J} \to \overline{J} \leftarrow \overline{J}} (\overline{J} \to \overline{H}).$$

Sketch of proof. By Lemma 4.10 or 4.12, there exists exactly one rule $\rho \in \mathscr{R}$ with L as a left-hand side that allows to derive transitions t_1 and t_2 – it is indeed the same rule for both. Let $\mathtt{D}$ be the composition diagram of the DPOBC-diagrams justifying the transitions.

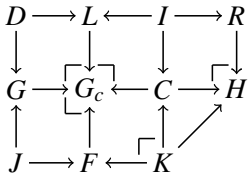
It is first shown that $\overline{G} \equiv \overline{G}_c$. Since the upper and lower left squares of $\mathtt{D}$ are pushouts we can infer that $\overline{D} \equiv L$ and $\overline{J} \equiv \overline{F}$. Finally, since no vertex is deleted (see Definition 4.7), we have $\overline{J} \to \overline{C}$ and thus $\overline{K} \equiv \overline{J}$.

So $\mathtt{D}$ is a BC-diagram of a τ-reaction from $\overline{J} \to \overline{G}$ to $\overline{J} \to \overline{H}$.

□

In fact, the main property that we have used is the following.

Definition 4.14 (Complementarity of Actions)**.** A graph transformation systems satisfies *Complementarity of Actions* if for each transition $(J \to G) \xrightarrow{J \to F \leftarrow K} (K \to H)$ there is a unique rule $L \leftarrow I \to R$ such that there exists a DPOBC-diagram as shown to the right.



In this situation, we can effectively determine if two transitions are τ-compatible. Thus we can derive a counterpart of the communication rule of CCS. Hence, if a graph transformation systems satisfies Complementarity of Actions then a rule of the following form is derivable in SOSBC.

$$\frac{t = (\overline{J} \to G) \xrightarrow{\overline{J} \to \overline{F} \leftarrow \overline{K}} (\overline{K} \to H) \qquad t' = (\overline{J} \to G') \xrightarrow{\overline{J} \to \overline{F}' \leftarrow \overline{K}'} (\overline{K}' \to H')}{(\overline{J} \to \overline{G}) \xrightarrow{\overline{J} \to \overline{J} \leftarrow \overline{J}} (\overline{J} \to \overline{H})} \quad t \text{ and } t' \ \tau\text{-compatible}$$

In other words, in a graph transformation system with Complementarity of Actions we can apply the results of [2] to obtain a counterpart to the communication rule.

5 Related and Future work

On a very general level, the present work is meant to strengthen the conceptual similarity of graph transformation systems and process calculi; thus it is part of a high-level research program that has been the theme of a Dagstuhl Seminar in 2005 [9]. In this wide field, structural operational semantics is occasionally considered as an instance of the tile model (see [8] for an overview). With this interpretation, SOS has served as motivation for work on operational semantics of graph transformation systems (e.g. [5]).

A new perspective on operational semantics, namely the "automatic" generation of labeled transition semantics from reaction rules, has been provided by the seminal work of Leifer and Milner [13] and its successors [16, 7]; as an example application, we want to mention the "canonical" operational semantics for the ambient calculus [15]. The main point of the latter work is the focus on the "properly" *inductive* definition of structural operational semantics. To the best of our knowledge, there is no recent work on the operational semantics of graph transformation systems that provides a general method for the inductive definition of *operational semantics*. This is not to be confused with the inductive definition of graphical encodings of process calculi on (global) states.

With this narrower perspective on techniques for the "automatic" generation of LTSs, we want to mention that some ideas of our three layer semantics in Section 3 can already be found in [3], where all rules of the definition of the labelled transition semantics have at most one premise. This is in contrast to the work of [15] where the labelled transition semantics is derived from two smaller subsystems: the process view and the context view; the subsystems are combined to obtain the operational semantics. The latter work is term based and it manipulates complete subterms of processes using the lambda calculus in the meta-language. We conjecture that the use of this abstraction mechanism is due to the term structure of processes.

Concerning future work, the first extension of the theory concerns more general (hierarchical) graph-like structures as captured by adhesive categories [10] and their generalizations (e.g. [4]). Moreover, as an orthogonal development, we plan to consider the case of more general rules that are allowed to have an arbitrary (graph) morphism on the right hand side; moreover, also states are arbitrary morphisms. The general rule format is important to model substitution in name passing calculi while arbitrary graph morphisms as states yield more natural representations of (multi-wire) interaction nets. The main challenge is the quest for more general sufficient conditions that allow for non-trivial compositions of labelled transitions, which can be seen as a general counterpart of the CCS communication rule.

6 Conclusion

We have reformulated the BC technique as the SOSBC-system in Table 1 to make a general analogy to the SOS-rules for CCS. There is no need for a counterpart of the communication rule. We conjecture that this is due to the "flat" structure of graphs as opposed to the tree structure of CCS-terms.

The main contribution concerns questions about the derivability of a counterpart of the communication rule. First, we give an example, which illustrates that the derivability of such a rule is non-trivial; however, it is derivable if the relevant graph transformation system satisfies *Complementarity of Actions*. We have given two classes of examples that satisfy this requirement, namely hyper-graphs with unique partners and simply wired hyper-graphs. This is a first step towards a "properly" inductive definition of structural operational semantics for graph transformation systems.

Acknowledgements We would like to thank Barbara König, Filippo Bonchi and Paolo Baldan for providing us

with drafts and ideas about a more general research program on compositionality in graph transformation. We are also grateful for the constructive criticism and the helpful comments of the anonymous referees.

References

[1] V. Alexiev (1999): *Non-deterministic interaction nets*. Ph.D. thesis, University of Alberta, Edmonton, Alta., Canada.

[2] P. Baldan, H. Ehrig & B. König (2006): *Composition and Decomposition of DPO Transformations with Borrowed Context*. In: *Proc. of ICGT '06 (International Conference on Graph Transformation)*, Springer, pp. 153–167, doi:10.1007/11841883_12. LNCS 4178.

[3] F. Bonchi, F. Gadducci & G. V. Monreale (2009): *Labelled transitions for mobile ambients (as synthesized via a graphical encoding)*. *Electronic Notes in Theoretical Computer Science* 242(1), pp. 73–98, doi:10.1016/j.entcs.2009.06.014.

[4] B. Braatz, H. Ehrig, G. Karsten & U. Golas (2010): *Finitary M-adhesive categories*. In: *Graph Transformations: 5th International Conference, ICGT 2010, Twente, the Netherlands, September 27–October 2, 2010, Proceedings*, Springer-Verlag, pp. 234–249, doi:10.1007/978-3-642-15928-2_16.

[5] Andrea Corradini, Reiko Heckel & Ugo Montanari (2000): *Graphical Operational Semantics*. In: *ICALP Satellite Workshops*, pp. 411–418.

[6] T. Ehrhard & L. Regnier (2006): *Differential interaction nets*. *Theoretical Computer Science* 364(2), pp. 166–195, doi:10.1016/j.tcs.2006.08.003.

[7] H. Ehrig & B. König (2006): *Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts*. *Mathematical Structures in Computer Science* 16(6), pp. 1133–1163, doi:10.1017/S096012950600569X.

[8] F. Gadducci & U. Montanari (2000): *The tile model*. In Gordon D. Plotkin, Colin Stirling & Mads Tofte, editors: *Proof, Language, and Interaction*, The MIT Press, pp. 133–166.

[9] B. König, U. Montanari & P. Gardner, editors (2005): *04241 Abstracts Collection*. *Dagstuhl Seminar Proceedings* 04241, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany. Available at `http://drops.dagstuhl.de/opus/volltexte/2005/27`.

[10] S. Lack & P. Sobociński (2005): *Adhesive and quasiadhesive categories*. *RAIRO - Theoretical Informatics and Applications* 39(2), pp. 522–546, doi:10.1051/ita:2005028.

[11] Y. Lafont (1990): *Interaction nets*. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, ACM, New York, NY, USA, pp. 95–108, doi:10.1145/96709.96718.

[12] Y. Lafont (1995): *From proof-nets to interaction nets*. In: *Proceedings of the workshop on Advances in linear logic*, Cambridge University Press, New York, NY, USA, pp. 225–247, doi:10.1017/CBO9780511629150.012.

[13] J. J. Leifer & R. Milner (2000): *Deriving Bisimulation Congruences for Reactive Systems*. In Catuscia Palamidessi, editor: *CONCUR*, *Lecture Notes in Computer Science* 1877, Springer, pp. 243–258, doi:10.1007/3-540-44618-4_19.

[14] D. Mazza (2006): *Interaction Nets: Semantics and Concurrent Extensions*. Ph.D. thesis, Université de la Méditerranée & Roma Tre.

[15] J. Rathke & P. Sobociński (2010): *Deriving structural labelled transitions for mobile ambients*. *Information and Computation* 208, pp. 1221–1242, doi:10.1016/j.ic.2010.06.001.

[16] V. Sassone & P. Sobociński (2003): *Deriving Bisimulation Congruences Using 2-categories*. *Nordic Journal of Computing* 10(2), pp. 163–183.

[17] A. Simpson (2004): *Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS*. *Journal of Logic and Algebraic Programming* 60–61, pp. 287–322, doi:10.1016/j.jlap.2004.03.004.

Computing approximations for graph transformation systems

Vincent Danos
School of Informatics
University of Edinburgh
vdanos@inf.ed.ac.uk

Tobias Heindel
School of Informatics
University of Edinburgh
theindel@inf.ed.ac.uk

Ricardo Honorato-Zimmer
School of Informatics
University of Edinburgh
r.honorato@sms.ed.ac.uk

Sandro Stucki
sandro.stucki@epfl.ch
Programming Methods Laboratory, EPFL

We describe a tool that can compute a differential equation for the mean occurrence counts of a fixed graph observable in a given stochastic graph transformation system. It is an open problem whether the function that gives the mean occurrence count of the graph motif at a fixed time on the positive real line is approximable to arbitrary precision. However, the tool allows to express common practices to approximate the function using mean-field and refined approximation techniques. In the long term, we plan an extension to stochastic bisimulation checking for graph transformation systems.

1 Introduction

In the context of dynamic systems where states have complex structure, graph transformation [2, 5] is a natural candidate for a general formalism to describe all possible transitions of systems. Graph transformation is a particularly useful specification formalism if all possible transitions between system states fall into a finite number of transition classes such that each transition class has a common principle of a local structural modification that can be captured by a single *rule* of graph transformation.

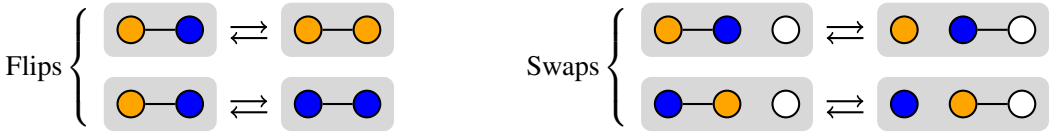
Graph transformation systems in which rules are equipped with rates gives rise to a continuous-time Markov chain [8]. Intuitively, transitions between states have now a propensity to take place in an infinitesimal small time step; formally, they define the infinitesimal generator of a Markov chain which has isomorphism classes of finite graphs as states.

The central topic of the paper is the calculation of the mean occurrence count of certain graphs of interest in function of time in such a stochastically evolving graph. A classic example that comes to mind is the count of triangles in a random graph; for stochastic graph transformation systems one might thus ask how their expected numbers change over time. In general, given a stochastically evolving graph (specified by a stochastic graph transformation system), the question that we seek to answer is how many occurrences of a certain fixed graph, the *observable graph pattern*, can one expect to have at a given time in the future. It is an open problem whether this function can be computed to arbitrary precision in general. We present a tool that allows to compute (approximations to) this continuous-time behaviour by deriving an ordinary differential equation (ODE) for the rate of change of the mean occurrence count of any graph motif. The procedure has been informally described in [3] and we provide an abriged version in §3. It is a well-known problem to give guarantees for the quality of the approximations. "Analytical approximation approaches vary in their complexity, and there is usually an associated trade-off in accuracy (as measured, for example, by comparing the prediction of the theory with a large-scale Monte Carlo simulation of the dynamics)" [11, p. 18]. Roughly, the problem is that the function that counts the number of occurrences of a graph does not have any bound *a priori*; we can nevertheless derive

Submitted to:
MeMo 2015

fundamental equations for variances of counting functions and sometimes even get precise solutions. The tool has the form of a Scala library named *graph-rewriting* and is available for download from `https://github.com/rhz/graph-rewriting/`.

We use the voter model treated in Ref. [3] as running example and show how to obtain its results automatically by the tool. In this model, nodes can be in either of two states, 'red' or 'blue', and there are two types of rewriting rules, so-called 'flips' and 'swaps':



White nodes are to be thought colourless, i.e. they represent nodes that can be in any of the two states. White nodes represent nodes that can be in any of the two states. Typically node colours are interpreted as people's opinions and thus these rules attempt to minimise disparity among acquaintances. These rules show an interesting behaviour that has been studied in depth in Ref. [4].

One question Ref. [4] sets out to answer is what happens in the long run. Clearly, if the graph reaches a state where everyone has the same opinion, no further rule applications are possible. Also, if the graph disconnects (due to the application of a swap), as soon as every connected component becomes monochrome the system halts. What will be the number of red and blue nodes in these halted states in average? Of course, this will depend, among other factors, on the likelihood of each flip with respect to one another and the fraction of red and blue nodes at the start. Will the network split before one colour wins over the other? Since the only way to split the network is by applying a swap, the likelihood of these two rules with respect to the flips will be crucial. To answer these questions quantitatively, we equip our rules with rates and compute the derivative of the number of red (or blue) nodes over time. In the running example, the rates are k_{01}, k_{10} for the flips from red (0) to blue (1) and from blue (1) to red (0), respectively; and k_0, k_1 are the rates for the swaps.

2 Graph transformation

Rules are like productions of Chomsky grammars, according to the analogy from the first paper of algebraic graph transformation [7]; in particular they have a left- and right-hand side. The first obvious difference are the kinds of structures being rewritten, i.e. graphs vs. strings. However, one subtle but important difference that the analogy does sweep under the carpet is the fact that applications of graph transformation rules always involve a single occurrence of the left- and right-hand side. For each rule application we always know where and how the rule is applied. This is essential for the definition of the continuous-time stochastic semantics of graph transformation.

A rule of transformation for any general type of structure consists of a left-hand side L, a right-hand side R, and a pair of partial maps from nodes and edges of L to those of R such that they jointly preserve the structure of the left hand side. as in single pushout graph transformation [10]. The nodes and edges that have an image along these partial maps are preserved by the rule, although their label might change. Instead, the nodes and edges in the left-hand side that do not have an image are destroyed, while those in the right-hand side that do not have a pre-image are created by an application of the rule. Note that the application of a rule always is relative to an occurrence of the left-hand side L in a graph G, which is formalised by an injective (total) graph morphism from L to G.

2.1 Graphs

The concrete graphs that are handled by the tool are node- and edge-labelled directed multi-graphs (although the general method applies to several other graph-like structures). Directed multi-graphs definition is as usual. The labelling of nodes and edges is formalised by a partial function from nodes (resp. edges) to colours for each state of the system, thus equipping graphs with a partial labelling of nodes (resp. edges). Colourless nodes are then those that are not in the domain of definition of the labelling function. The ocurrence count of a graph G is written as $[G]$ and $G+H$ denotes the disjoint union of G and H.

In the `graph-rewriting` library, graphs are parametric in the type of nodes, edges, and labels they contain. In particular, the type of nodes and labels can be anything. Edges however can only be an instance of a class implemeting the `DiEdgeLike` interface. Any class that defines a `source` and `target` method can implement this interface. For multi-edges an `id` is needed to differentiate between edges that have the same `source` and `target`. The library comes with a default class for multi-edges, `IdDiEdge`, and graphs must be comprised of edges of this type to be fed to the ODE generation algorithm.

The example introduced in the previous section uses undirected graphs instead of directed graphs. To encode the model, we could split each rule into two versions, one for each direction (if the rules had more than one edge it would be a combinatorial expansion though!). Here we take a simpler approach instead: edges always go from red to blue.

A multi-edge can be created by using `IdDiEdge`'s constructor, e.g. the expression `IdDiEdge(0,"u","v")` will create an edge from node "u" to "v" with an `id` of type `Int` and value `0`. Also, an edge can be created by using `~~>`, as in `"u"~~>"v"`. This will implicitly define an `id` for that edge by means of a global counter that will not generate the same `id` twice but could produce an `id` that has been already used by the user.

Graphs are mutable and so can be constructed progressively by adding nodes, edges and labels to them. When a new graph is instantiated, an initial set of nodes, edges and labels can be defined. For example, `Graph("u"->"l1","v"->"l2")("u"~~>"v")` will create a graph with two nodes "u" and "v" with labels "l1" and "l2" respectively and an unlabelled edge from "u" to "v". More specifically, the `Graph` constructor takes as a first set of parameters any number of nodes with an optional label each, indicated by an arrow (`->`). Then it takes a second set of parameters for edges and their optional labels in the same fashion. Note that some nodes and edges can be left unlabelled while others are labelled. For instance, the left-hand side of the first swap rule may be constructed by the following expression: `Graph("u"->"red","v"->"blue","w")("u"~~>"v")`.

Whenever we want to create a Graph that does not have edges or nodes or labels, we must specify a type for them. Otherwise Scala's type inference will assign them a `Nothing` type. There are 4 type parameters in total for `Graph`'s constructor: the node type, node label type, edge type, and edge label type. The first two are specified before the first set of parameters and the other two right before the second set of parameters, as in `Graph[String,String]("u")[IdDiEdge[Int, String],String]()`. Repeating the type parameters again every time we would like to instantiate a `Graph` can rapidly become unwieldy and in models usually all `Graphs` will have the same type signature. For this reason the library let us create custom constructors with specific type parameters defined beforehand by using `Graph`'s `withType` method, e.g.

```
val G = Graph.withType[String,String,IdDiEdge[Int,String],String]
val oneNodeGraph = G("u")()
```

2.2 Rules

As mentioned earlier, rules are described by two graphs (L and R) and a pair of partial maps between them for nodes and edges. The library uses Scala's `Map` to define the latter. Additionally, rules are equipped with a `Rate`, which are comprised of a name and a value and constructed as `Rate(‘‘rate name’’, rateValue)` with `rateValue` a `Double`, or implicitly from a `String`, as in the following example (with an implicit value of `1.0`). Consider the first swap rule in our example. One way to construct this rule would be:

```
val lhs = Graph("u"->"red","v"->"blue","w")("u"~~>"v")
val rhs = Graph("u"->"red","v"->"blue","w")("v"~~>"w")
val swap0 = Rule(lhs, rhs, Map("u"->"u","v"->"v","w"->"w"),
                 Map(), "k0")
```

Where `"k0"` is the name for the rate. In the generated ODEs it will appear by this name instead of its numerical value. This allows an easier interpretation of where the terms of each ODE come from. In the definition of this `Rule` all nodes are preserved while no edge is, since the partial map for edges is empty. Note that, when mapping edges, attention has to be paid to the `ids` of edges, not just their `source` and `target`. In our example, the flip rule preserves the edge in its left-hand side and can be constructed using the following code:

```
val e = "u"~~>"v"
val lhs = Graph("u"->"red","v"->"blue")(e)
val rhs = Graph("u"->"red","v"->"red")(e)
val flip0 = Rule(lhs, rhs, Map("u"->"u","v"->"v"),
                 Map(e->e), "k01")
```

The complete code needed to define all rules can be found in Appendix A.

3 ODE generation

The ODE generation algorith depends heavily on a graph construction called *minimal glueings* in Ref. [3] and related to local co-products. This construction allows us to characterise all pair of matches of any two graphs onto a common target graph into a unique family among finitely many. Each family represents a way in which the two graphs can overlap. Intuitively, we use it to enumerate all possible ways in which the application of a rule could create or destroy instances of the desired observables. It is worth noting that, although finite, there may be exponentially many minimal glueings (i.e. families) for a pair of graphs!

Call $m(A,B)$ the set of minimal glueings of A and B. The algorithm proceeds as follows, given an observable G: 1) for each rule left-hand side L, compute $m(G,L)$; 2) for each rule right-hand side R, compute $m(G,R)$ and apply the inverse rule to each element of $m(G,R)$ – call this set $m'(G,R)$; 3) generate an ODE for $[G]$ of the form:

$$\frac{d}{dt}[G] = \sum_{(L \rightharpoonup R,k)\in\mathscr{R}} k \left(\sum_{H\in m'(G,R)} [H] - \sum_{F\in m(G,L)} [F] \right)$$

4) We repeat the procedure for the new observables $[H]$ and $[F]$.

3.1 Equations

We have just seen how to build flip and swap rules and so we are ready to use our algorithm to obtain a system of differential equations that can describe the average count of an observable in time. To do this we use the `generateMeanODEs` method in `moments`. This method accepts 3 mandatory parameters, namely, the maximum number of ODEs to be discovered, a list of rules and a list of observables. Optionally we can provide so-called 'transformers' which are explained in the next section. As output, this method returns a list of equations. We can use `ODEPrinter` to print them to the standard output or to save them in an Octave script that will integrate the system of differential equations.

To answer one of our initial questions about the model – namely how many red (or blue) nodes there are in the halted state – let us consider the following simple observable: a single red node.

```
val redNode = G("u" -> "red")()
val equations = meanfield.mfa(2,
  List(flip0,flip1,swap0r,swap0b,swap1r,swap1b), List(redNode))
```

Here we use the custom `Graph` constructor defined above, `G`. To print these `equations`, we use `ODEPrinter(equations).print`. The output, rendered graphically, is:

$$\frac{d}{dt}\left[\;\right] = (k_{10}-k_{01})\left(\left[\;\right]+\left[\;\right]\right) \tag{1}$$

$$\begin{aligned}\frac{d}{dt}\left[\;\right] = {} & -k_{10}\left[\;\right]-k_{01}\left[\;\right]+k_{10}\left[\;\right]+k_{01}\left[\;\right] \\ & -k_{10}\left[\;\right]-k_{01}\left[\;\right]+k_{10}\left[\;\right]+k_{01}\left[\;\right] \\ & -(k_{10}+k_{01})\left(\left[\;\right]+\left[\;\right]+\left[\;\right]\right)-(k_1+k_0)\left[\;\right]\end{aligned} \tag{2}$$

If we were to ask for the ODEs of these observables, we would get bigger observables and from the ODEs of these even bigger observables in a process that would never terminate. However, we are interested in obtaining a system of differential equations that is closed. To this end, we have to make some approximations which can be integrated algorithmically by using 'transformers'.

The reader might wonder why the second ODE listed above has a term for the graph with a red and a blue node with two edges between them. This comes from the fact that whenever we apply a flip to a pair of nodes that have more than one edge between them, more than one instance of the red-to-blue pattern is deleted *at once*. It is indeed sufficient to look at each pair of multi-edges to account for all those deletions.

3.2 Transformers: approximations and simplifications

A transformer is any function defined on `Graphs` that returns a graph monomial or nothing (i.e. an `Option[Mn[N,NL,E,EL]]` with `N`, `NL`, `E`, and `EL` the `Graph`'s type parameters). A transformer is applied whenever a new observable is discovered by the ODE generation mechanism. If it returns a graph monomial, an algebraic (instead of differential) equation is generated for that observable, equating the observable to the returned monomial. Here is where the user is given the freedom to equate expressions that are not actually equal and thus introduce approximations. It is worth noting that in the abscence of transformers, all ODEs discovered by the algorithm are exact. If the transformer returns nothing, then the algorithm will compute an ODE for the observable (as long as the maximum number of ODEs to be generated hasn't been reached).

The first transformer that we introduce in our example will help us to break down disconnected graph observables into their constituent connected components, keeping the size of some graph observables bounded. Formally, this amounts to say that the expected value of the ocurrence counts of a disconnected graph observable (given the time-dependent probability distribution p on the the state space), $\mathrm{E}_p([G])$ with $G = A + B + ... + C$, is approximately $\mathrm{E}_p([A]) \cdot \mathrm{E}_p([B]) \cdot ... \cdot \mathrm{E}_p([C])$. This approximation is akin to that used in the theory of Petri nets for computing rate equations, e.g. that $\frac{d}{dt}\mathrm{E}_p([A]) = \mathrm{E}_p([2A]) \simeq \mathrm{E}_p([A])^2$ in the system comprised of the single reaction $2A \to 3A$. It is based on the assumption of independence between observables. In the following code snippets, we assume type `N` is `String`, `E` is `IdDiEdge[Int,N]`, and `NL`, `EL` are any types.

```
def splitConnectedComponents(g: Graph[N,NL,E,EL]): Option[Mn[N,NL,E,EL]] =
  if (g.isConnected) None else Some(Mn(g.components))
```

This is indeed a general transformation and thus it is supplied as part of the library. This transformer will return `None` when the graph is connected, meaning that connected graph observables will not be approximated by it.

The second approximation introduced is the so-called "pair approximation", which in the most general case entails approximating a mean ocurrence count of a graph G as the product of two overlapping subgraphs G_1, G_2 such that $G_1 \cup G_2 = G$, divided by their intersection I, i.e. $\mathrm{E}_p([G]) \simeq \mathrm{E}_p([G_1])\mathrm{E}_p([G_2])/\mathrm{E}_p([I])$. This approximation assumes conditional independence of the overlapping subgraphs.

```
def pairApprox(g: Graph[N,NL,E,EL]): Option[Mn[N,NL,E,EL]] =
  if (g.nodes.size == 3 && g.edges.size == 2 && g.isConnected) {
    val List(e1, e2) = g.edges.toList
    val intersection = e1.nodes &~ (e1.nodes &~ e2.nodes)
    Mn(g.inducedSubgraph(e1.nodes)) *
       g.inducedSubgraph(e2.nodes) /
       g.inducedSubgraph(intersection)
  } else None
```

Where `&~` is Scala's set difference operator. The code presented here performs a "pair approximation" on graphs with 3 nodes and 2 edges like the ones obtained in Eq. 2, splitting them into two graphs with 2 nodes and 1 edge each, and the intersection being the node in the middle.

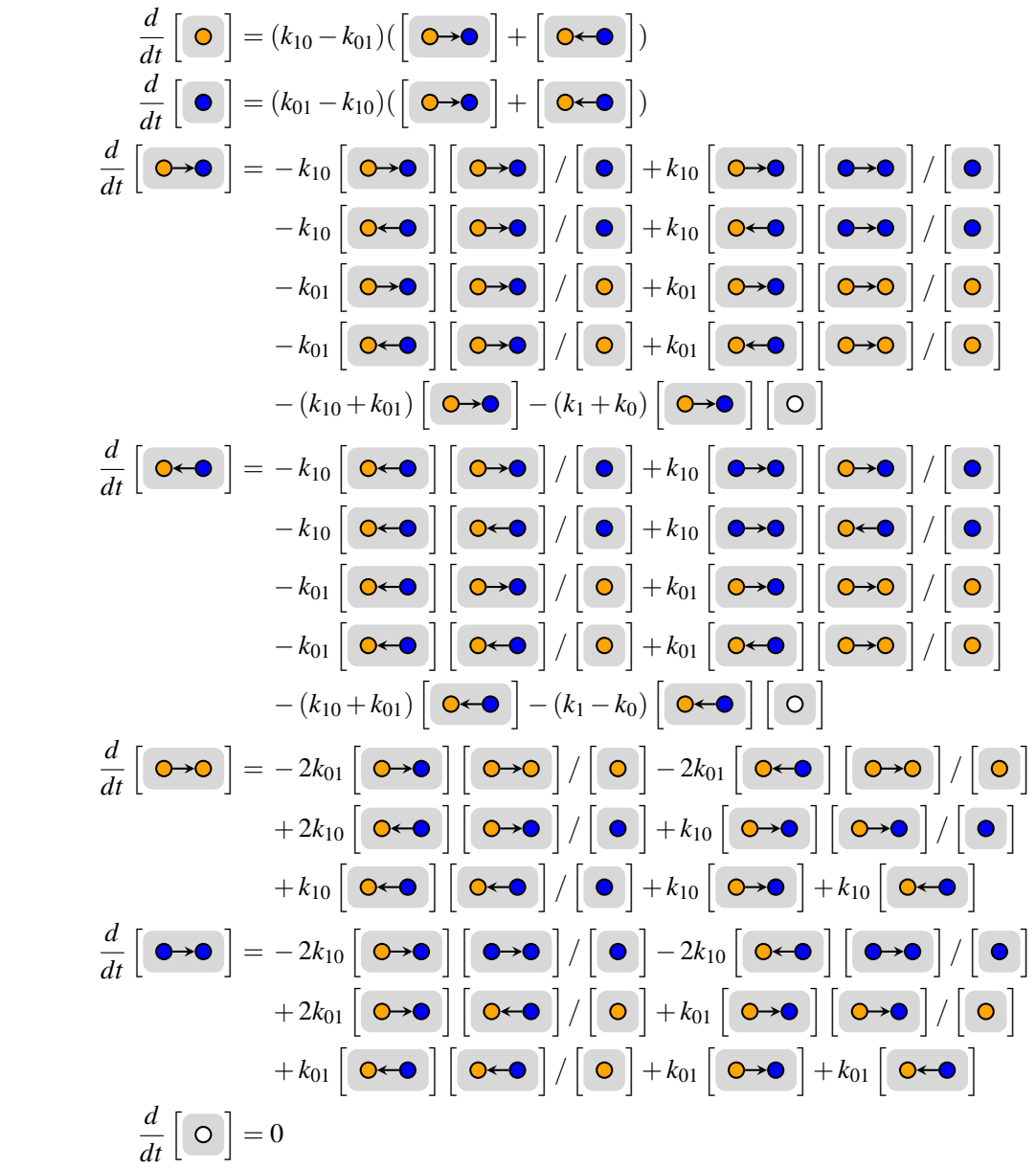
Finally, the only observables left that blow up the expansion are the terms with multi-edges between two nodes. However, if the initial state is a sparse graph, a multi-edge is highly unlikely to ever occur. Hence we approximate this observable's average ocurrence count as zero.

```
def noParallelEdges(g: Graph[N,NL,E,EL]): Option[Mn[N,NL,E,EL]] =
  if (g.nodes.size == 2 && g.edges.size == 2) Some(Mn.zero)
  else None
```

4 Example results

In this section we show the results obtained from running the model in our example as presented in Appendix A. Please note that we have manually translated the textual output notation given by the tool

into the graphical notation used throughout this paper.



Note that the algorithm discovered that the number of nodes with any colour is invariant. By generating an Octave script using `saveAsOctave` method in `ODEPrinter`, the system of differential equations can be integrated numerically. In particular, the solution to the set of ODEs above has been computed and is shown in Fig. 1.

4.1 Variance and other moments

In addition to the mean number of red nodes ($\mathrm{E}_p([0])$), we can compute what the variance of this number is by using the formula $\mathrm{V}([0]) = \mathrm{E}_p([0]^2) - \mathrm{E}_p([0])^2$. Minimal glueings let us express $[0]^2$ as a sum of

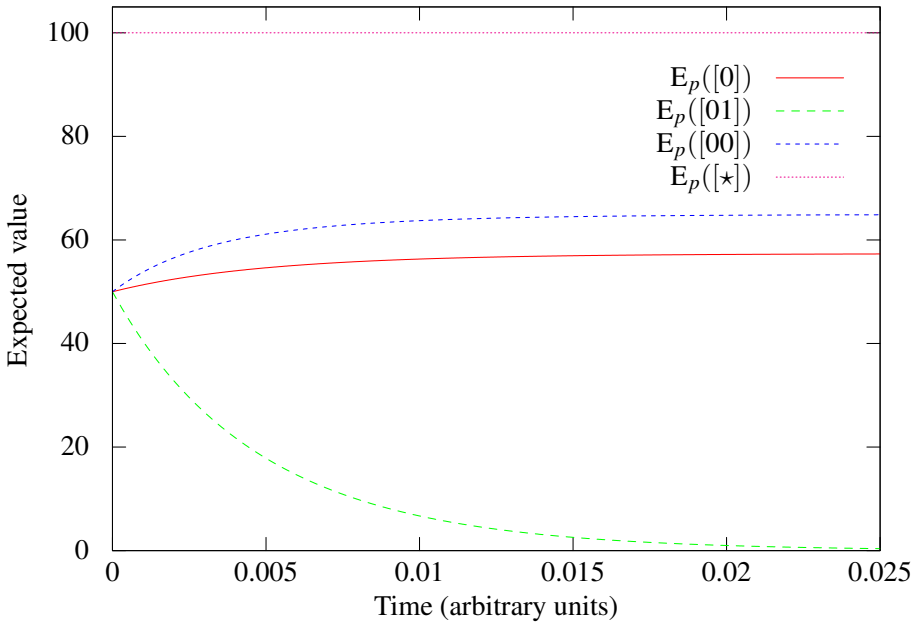


Figure 1: Solution to the system of ordinary differential equations given the following initial conditions: $[0] = 50, [1] = 50, [01] = 50, [10] = 50, [00] = 50, [11] = 50$, and $[\star] = 100$ with $\star$ the graph consisting of a single colourless node and rates $k_{10} = 15$, $k_{01} = 0.1$, $k_0 = 1$, and $k_1 = 1$.

observables: $[0]^2 = [0+0] + [0]$. For this reason, we compute an ODE for $\mathrm{E}_p([0+0])$ without using any transformers (`splitConnectedComponents` would transform $[0+0]$ back into $[0]^2$ and the other two are irrelevant in this case).

$$\frac{d}{dt}\left[\;\circ\;\circ\;\right] = 2(k_{10}-k_{01})\left(\left[\;\circ\;\circ\!\!\rightarrow\!\!\bullet\;\right] + \left[\;\circ\;\circ\!\!\leftarrow\!\!\bullet\;\right]\right) + 2k_{10}\left(\left[\;\circ\!\!\rightarrow\!\!\bullet\;\right] + \left[\;\circ\!\!\leftarrow\!\!\bullet\;\right]\right) \qquad (3)$$

Formulas exist that relate the value of any moment to lower moments. Hence these higher moments can be computed in a similar fashion to the one shown here.

5 Related and future work

A complete list of related work is beyond the scope of the paper: We expect a large number of concrete models that have been studied extensively in application-specific, *ad hoc* terminology to be captured equally well by graph transformation. We refer the reader to Ref. [11] for a recent overview on dynamical systems on networks that are of particular interest to the authors and gives references for a relevant part of the immense body of literature; in particular, the type of approximations that we expect to be used as transformers in our tool are related to those listed under the heading of Mean-Field Theories, Pair Approximations, and Higher-Order Approximations in the latter work.

A modelling formalism that is closely related to the algebraic approach to graph transformation are process calculi. Before we sketch possible future research on the connection to this field, especially recent work on stochastic process calculi such as [1], we point out some important differences. The

terminology of *observable* is inspired by (classical) physics, chemistry and biology. In particular, graph observables are *a priori* unrelated to the notion of observable actions from the process calculi literature and there is no direct link to observational equivalences; also, the relation to observability in control theory is unclear at best. Similarly, the question of whether it is suitable to consider a certain graph as observable cannot be answered from the set of rules and the initial state alone; it is necessary to have additional information from the specific application at hand. In examples from biology, such as protein translation as modelled by the TASEP (see Ref. [13] for a tutorial), a graph would be suitable for observation if the evolution of its mean occurrence counts would have a corresponding set of experiments that allow to falsify or validate the model.

For the possible relations of the present paper to the process calculus literature that do exist, note first that rule applications of graph transformation correspond to single step reductions in process calculi. The ground-breaking observation of Leifer and Milner [9] was that one can automatically derive labelled transition systems from reduction semantics (for process calculi). This idea has been successfully transferred to graph transformation [6]. We plan to use techniques from Ref. [6] to canonically associate labelled Markov processes to graph transformation systems. We expect that analogues to the semantics of CCS as labelled Markov processes given in [1] can be derived without complications; moreover, the resulting labelled Markov processes might be meaningful for complex networks modelled by graph transformation (while we would be rather surprised if such interactive semantics have applications in biology or chemistry). In a slightly different direction and as a long term goal, we want to explore whether the data structures and operations on graphs that are implemented in the tool can be re-used for checking (approximate) stochastic bisimulation of graph transformation systems.

References

[1] Luca Cardelli & Radu Mardare (2014): *The Measurable Space of Stochastic Processes*. *Fundam. Inform.* 131(3-4), pp. 351–371, doi:10.3233/FI-2014-1019. Available at `http://dx.doi.org/10.3233/FI-2014-1019`.

[2] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel & Michael Löwe (1997): *Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach*. In Rozenberg [12], pp. 163–246.

[3] Vincent Danos, Tobias Heindel, Ricardo Honorato-Zimmer & Sandro Stucki (2014): *Approximations for Stochastic Graph Rewriting*. In: *ICFEM*, pp. 1–10, doi:10.1007/978-3-319-11737-9_1. Available at `http://dx.doi.org/10.1007/978-3-319-11737-9_1`.

[4] Richard Durrett, James P Gleeson, Alun L Lloyd, Peter J Mucha, Feng Shi, David Sivakoff, Joshua ES Socolar & Chris Varghese (2012): *Graph fission in an evolving voter model*. *Proceedings of the National Academy of Sciences* 109(10), pp. 3682–3687.

[5] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner & Andrea Corradini (1997): *Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach*. In Rozenberg [12], pp. 247–312.

[6] Hartmut Ehrig & Barbara König (2006): *Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts*. *Mathematical Structures in Computer Science* 16(6), pp. 1133–1163, doi:10.1017/S096012950600569X. Available at `http://dx.doi.org/10.1017/S096012950600569X`.

[7] Hartmut Ehrig, Michael Pfender & Hans Jürgen Schneider (1973): *Graph-Grammars: An Algebraic Approach*. In: *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, IEEE Computer Society, pp. 167–180, doi:10.1109/SWAT.1973.11. Available at `http://dx.doi.org/10.1109/SWAT.1973.11`.

[8] Reiko Heckel, Georgios Lajios & Sebastian Menge (2006): *Stochastic Graph Transformation Systems*. *Fundamenta Informaticae* 74(1), pp. 63–84. Available at http://iospress.metapress.com/content/c7ha18g96nbm7g2e/.

[9] James J. Leifer & Robin Milner (2000): *Deriving Bisimulation Congruences for Reactive Systems*. In Catuscia Palamidessi, editor: *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, *Lecture Notes in Computer Science* 1877, Springer, pp. 243–258, doi:10.1007/3-540-44618-4_19. Available at http://dx.doi.org/10.1007/3-540-44618-4_19.

[10] Michael Löwe (1993): *Algebraic Approach to Single-Pushout Graph Transformation*. *Theor. Comput. Sci.* 109(1&2), pp. 181–224, doi:10.1016/0304-3975(93)90068-5. Available at http://dx.doi.org/10.1016/0304-3975(93)90068-5.

[11] Mason A. Porter & James P. Gleeson (2014): *Dynamical Systems on Networks: A Tutorial*. *CoRR* abs/1403.7663. Available at http://arxiv.org/abs/1403.7663.

[12] Grzegorz Rozenberg, editor (1997): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific.

[13] R.K.P. Zia, J.J. Dong & B. Schmittmann (2011): *Modeling Translation in Protein Synthesis with TASEP: A Tutorial and Recent Developments*. *Journal of Statistical Physics* 144(2), pp. 405–428, doi:10.1007/s10955-011-0183-1. Available at http://dx.doi.org/10.1007/s10955-011-0183-1.

A Voter model

```
import graph_rewriting._
import implicits._
import moments._ // this imports N=String and E=IdDiEdge[Int,N]
type NL = String
type EL = String
val G = Graph.withType[N,NL,E,EL]
// first flip
val e = "u"~~>"v"
val rb = G("u"->"red","v"->"blue")(e)
val br = G("u"->"blue","v"->"red")(e)
val bb = G("u"->"blue","v"->"blue")(e)
val flip0a = Rule(rb, bb, Map("u"->"u","v"->"v"),
  Map(e->e), "k01")
val flip0b = Rule(br, bb, Map("u"->"u","v"->"v"),
  Map(e->e), "k01")
// second flip
val rr = G("u"->"red","v"->"red")(e)
val flip1a = Rule(rb, rr, Map("u"->"u","v"->"v"),
  Map(e->e), "k10")
val flip1b = Rule(br, rr, Map("u"->"u","v"->"v"),
  Map(e->e), "k10")
// first swap (blue rewire)
val rbw1 = G("u"->"red","v"->"blue","w")("u"~~>"v")
val rbw2 = G("u"->"red","v"->"blue","w")("w"~~>"v")
val swap0a = Rule(rbw1, rbw2, Map("u"->"u","v"->"v","w"->"w"),
```

```
  Map(), "k0")
val brw1 = G("u"->"blue","v"->"red","w")("u"~~>"v")
val brw2 = G("u"->"blue","v"->"red","w")("w"~~>"u")
val swap0b = Rule(brw1, brw2, Map("u"->"u","v"->"v","w"->"w"),
  Map(), "k0")
// second swap (red rewire)
val rbw3 = G("u"->"red","v"->"blue","w")("w"~~>"u")
val swap1a = Rule(rbw1, rbw3, Map("u"->"u","v"->"v","w"->"w"),
  Map(), "k1")
val brw3 = G("u"->"blue","v"->"red","w")("w"~~>"v")
val swap1b = Rule(brw1, brw3, Map("u"->"u","v"->"v","w"->"w"),
  Map(), "k1")
def pairApproximation(g: Graph[N,NL,E,EL]): Option[Mn[N,NL,E,EL]] =
  if (g.nodes.size == 3 && g.edges.size == 2 && g.isConnected) {
    val List(e1, e2) = g.edges.toList
    val intersection = e1.nodes &~ (e1.nodes &~ e2.nodes)
    Mn(g.inducedSubgraph(e1.nodes)) *
       g.inducedSubgraph(e2.nodes) /
       g.inducedSubgraph(intersection)
  } else None
def noParallelEdges(g: Graph[N,NL,E,EL]): Option[Mn[N,NL,E,EL]] =
  if (g.nodes.size == 2 && g.edges.size == 2) Some(Mn.zero) else None
val redNode = G("u" -> "red")()
val twoRedNodes = G("u" -> "red", "v" -> "red")()
val odes = generateMeanODEs(10,
  List(flip0a,flip0b,flip1a,flip1b,swap0a,swap0b,swap1a,swap1b),
  List(redNode),
  splitConnectedComponents[N,NL,E,EL] _,
  pairApproximation _,
  noParallelEdges _)
ODEPrinter(odes).print
println()
val varianceODE = generateMeanODEs(1,
  List(flip0a,flip0b,flip1a,flip1b,swap0a,swap0b,swap1a,swap1b),
  List(twoRedNodes))
ODEPrinter(varianceODE).print
```

Modal Logics for Nominal Transition Systems

Joachim Parrow Johannes Borgström Lars-Henrik Eriksson
Ramūnas Gutkovas Tjark Weber

We define a uniform semantic substrate for a wide variety of process calculi where states and action labels can be from arbitrary nominal sets. A Hennessy-Milner logic for these systems is introduced, and proved adequate for bisimulation equivalence. We show how to treat different bisimulation variants such as early, late and open in a systematic way, and make substantial comparisons with related work. The main definitions and theorems have been formalized in Nominal Isabelle.

1 Introduction

Transition systems. Transition systems are ubiquitous in models of computing, and specifications to say what may and must happen during executions are often formulated in a modal logic. There is a plethora of different versions of both transition systems and logics, including a variety of higher-level constructs such as updatable data structures, new name generation, alias generation, dynamic topologies for parallel components etc. In this paper we formulate a general framework where such aspects can be treated uniformly, and define accompanying modal logics which are adequate for bisimulation. This is related to, but independent of, our earlier work on psi-calculi [4], which proposes a particular syntax for defining behaviours. The present paper does not depend on any such language, and provides general results for a large class of transition systems.

In any transition system there is a set of *states* $P, Q, \ldots$ representing the configurations a system can reach, and a relation telling how a computation can move between them. Many formalisms, for example all process algebras, define languages for expressing states, but in the present paper we shall make no assumptions about any such syntax.

In systems describing communicating parallel processes the transitions are labelled with *actions* α, β, representing the externally observable effect of the transition. A transition $P \xrightarrow{\alpha} P'$ thus says that in state P the execution can progress to P' while conducting the action α, which is visible to the rest of the world. For example, in CCS these actions are atomic and partitioned into output and input communications. In value-passing calculi the actions can be more complicated, consisting of a channel designation and a value from some data structure to be sent along that channel.

Scope openings. With the advent of the pi-calculus [19] an important aspect of transitions was introduced: that of name generation and scope opening. The main idea is that names (i.e., atomic identifiers) can be scoped to represent local resources. They can also be transmitted in actions, to give a parallel entity access to this resource. In the monadic pi-calculus such an action is written $\overline{a}(\nu b)$, to mean that the local name b is exported along the channel a. These names can be subjected to alpha-conversion: if $P \xrightarrow{\overline{a}(\nu b)} P'$ and c is a fresh name then also $P \xrightarrow{\overline{a}(\nu c)} P'\{c/b\}$, where $P'\{c/b\}$ is P' with all bs replaced by cs. Making this idea fully formal is not entirely trivial and many papers gloss over it. In the polyadic pi-calculus several names can be exported in one action, and in psi-calculi arbitrary data structures may contain local names. In this paper we make no assumptions about how actions are expressed, and just assume that for any action α there is a finite set of names $\mathrm{bn}(\alpha)$, the *binding names*, representing exported

names. In our formalization we use nominal sets, an attractive theory to reason about objects depending on names on a high level and in a fully rigorous way.

State predicates. The final general components of our transition systems are the *state predicates* ranged over by φ, representing what can be concluded in a given state. For example state predicates can be equality tests of expressions, or connectivity between communication channels. We write $P \vdash \varphi$ to mean that in state P the state predicate φ holds.

A structure with states, transitions and state predicates as discussed above we call a *nominal transition system*.

Hennessy-Milner Logic. Modal logic has been used since the 1970s to describe how facts evolve through computation. We use the popular and general branching time logic known as Hennessy-Milner Logic [15] (HML). Here the idea is that an action modality $\langle\alpha\rangle$ expresses a possibility to perform an action α. If A is a formula then $\langle\alpha\rangle A$ says that it is possible to perform α and reach a state where A holds. With conjunction and negation this gives a powerful logic shown to be *adequate* for bisimulation equivalence: two processes satisfy the same formulas exactly if they are bisimilar. In the general case, conjunction must take an infinite number of operands when the transition systems have states with an infinite number of outgoing transitions. The fully formal treatment of this requires care in ensuring that such infinite conjunctions do not exhaust all names, leaving none available for alpha-conversion. All previous works that have considered this issue properly have used uniformly bounded conjunction, i.e. the set of all names in all conjuncts is finite.

Contributions. Our definition of nominal transition systems is very general since we leave open what the states, transitions and predicates are. The only requirement is that transitions satisfy alpha-conversion. A technically important point is that we do not assume the usual *name preservation principle*, that if $P \xrightarrow{\alpha} P'$ then the names occurring in P' must be a subset of those occurring in P and α. This means that the results are applicable to a wide range of calculi. For example, the pi-calculus represents a trivial instance where there are no state predicates. CCS represent an even more trivial instance where bn always returns the empty set. In the fusion calculus and the applied pi-calculus the state contains an environmental part which tells what expressions are equal to what. In the general framework of psi-calculi the states are processes with assertions describing their environments.

We define a modal logic with the $\langle\alpha\rangle$ operator that binds the names in $\mathrm{bn}(\alpha)$, and contains operators for state predicates. In this way we get a logic for an arbitrary nominal transition system such that logical equivalence coincides with bisimilarity. We also show how variants of the logic correspond to late, open and hyperbisimilarity in a uniform way. The main technical difficulty is to ensure that formulas and their alpha-equivalence classes throughout are finitely supported, i.e. only depend on a finite set of names, even in the presence of infinite conjunction. Instead of uniformly bounded conjunction we use the notion of finite support from nominal sets. This results in greater generality and expressiveness. For example, we can now define quantifiers and the next step modalities as derived operators.

Formalization. Our main definitions and theorems have been formalized in Nominal Isabelle [26]. This has required significant new ideas to represent data types with infinitary constructors like infinite conjunction and their alpha-equivalence classes. As a result we corrected several details in our formulations and proofs, and now have very high confidence in their correctness. The formalization effort has been substantial, but certainly less than half of the total effort, and we consider it a very worthwhile investment.

Exposition. In the following section we provide the necessary background on nominal sets. In Section 3 we present our main definitions and results on nominal transition systems and modal logics. In Section 4 we derive useful operators such as quantifiers and fixpoints, and indicate some practical uses. Section 5 shows how to treat variants of bisimilarity such as late and open in a uniform way, and in Section 6 we compare with related work and demonstrate how our framework can be applied to recover earlier results uniformly. Finally Section 7 concludes with some remarks on the formalization in Nominal Isabelle. All proofs are relegated to the Appendix.

2 Background on nominal sets

Nominal sets [24] is a general theory of objects which depend on names, and in particular formulates the notion of alpha-equivalence when names can be bound. The reader need not know nominal set theory to follow this paper, but some key definitions will make it easier to appreciate our work and we recapitulate them here.

We assume an infinitely countable multi-sorted set of atomic identifiers or *names* $\mathcal{N}$ ranged over by $a, b, \ldots$. The *permutations* is the group of name permutations that leave all but finitely many names invariant. The singleton permutation which swaps names a and b and has no other effect is written $(a\,b)$, and the identity permutation that swaps nothing is written id. Permutations are ranged over by π, π'. The effect of applying a permutation π to an object X is written $\pi \cdot X$. Formally, the permutation action $\cdot$ can be any operation that satisfies $\mathrm{id} \cdot X = X$ and $\pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X$, but a reader may comfortably think of $\pi \cdot X$ as the object obtained by permuting all names in X according to π.

A set of names N *supports* an object X if for all π that leave all members of N invariant it holds $\pi \cdot X = X$. In other words, a support N of X is such that names outside N do not matter to X. If X has a finite support then it also has a unique minimal support, written $\mathrm{supp}(X)$, intuitively consisting of exactly the names that matter to X. As an example the set of names textually occurring in a datatype element is the support of that element, and the set of free names is the support of the alpha equivalence class of the element. Note that in general, the support of a set is not the same as the union of the support of its members. An example is the set of all names; each element has itself as support, but the whole set has empty support since any permutation of the set yields the same set.

We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{supp}(X)$. The intuition is that if $a\#X$ then X does not depend on a in the sense that a can be replaced with any fresh name without affecting X. If A is a set of names we write $A\#X$ for $\forall a \in A \,.\, a\#X$.

A *nominal set* S is a set with a permutation action such that $X \in S \Rightarrow \pi \cdot X \in S$, and where each member $X \in S$ has finite support. A main point is that then each member has infinitely many fresh names available for alpha-conversion. Similarly, a set of names N supports a function f on a nominal set if for all π that leave N invariant it holds $\pi \cdot f(X) = f(\pi \cdot X)$, and similarly for relations and functions of higher arity. Thus we extend the notion of support to finitely supported functions and relations as the minimal finite support, and can derive general theorems such as $\mathrm{supp}(f(X)) \subseteq \mathrm{supp}(f) \cup \mathrm{supp}(X)$.

An object that has empty support we call *equivariant*. For example, a unary function f is equivariant if $\pi \cdot f(X) = f(\pi \cdot X)$ for all π, X. The intuition is that an equivariant object does not treat any name special.

3 Nominal transition systems and Hennessy-Milner logic

Definition 1. *A* nominal transition system *is characterized by the following*

- STATES*: A nominal set of* states *ranged over by* P, Q.
- PRED*: A nominal set of* state predicates *ranged over by* φ.
- *An equivariant binary relation* $\vdash$ *on* STATES *and* PRED. *We write* $P \vdash \varphi$ *to mean that in state* P *the state predicate* φ *holds.*
- ACT*: A nominal set of* actions *ranged over by* α.
- *An equivariant function* bn *from* ACT *to finite sets of names, which for each* α *returns a subset of* $\mathrm{supp}(\alpha)$*, called the* binding names.
- *An equivariant transition relation* $\rightarrow$ *on states and residuals. A residual is a pair of action and state. For* $\rightarrow (P,(\alpha,P'))$ *we write* $P \xrightarrow{\alpha} P'$. *The transition relation must satisfy alpha-conversion of residuals: If* $a \in \mathrm{bn}(\alpha)$, $b\#\alpha, P'$ *and* $P \xrightarrow{\alpha} P'$ *then also* $P \xrightarrow{(a\,b)\cdot\alpha} (a\,b)\cdot P'$.

Definition 2. *A* bisimulation *R is a symmetric binary relation on states in a nominal transition system satisfying the following two criteria: $R(P,Q)$ implies*

1. Static implication*: $P \vdash \varphi$ implies $Q \vdash \varphi$.*
2. Simulation*: For all α, P' such that $\mathrm{bn}(\alpha)\#Q$ there exist Q' such that if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ and $R(P',Q')$*

We write $P \dot{\sim} Q$ to mean that there exists a bisimulation R such that $R(P,Q)$.

Static implication means that bisimilar states must satisfy the same state predicates; this is reasonable since these can be tested by an observer. The simulation requirement is familiar from the pi-calculus.

Proposition 3. *$\dot{\sim}$ is an equivariant equivalence relation.*

The minimal HML for nominal transition systems is the following.

Definition 4. *The nominal set of formulas $\mathscr{A}$ ranged over by A is defined by induction as follows:*

$$A \quad ::= \quad \bigwedge_{i\in I} A_i \quad | \quad \neg A \quad | \quad \varphi \quad | \quad \langle\alpha\rangle A$$

Support and name permutation are defined as usual (permutation distributes over all formula constructors). In $\bigwedge_{i\in I} A_i$ it is assumed that the indexing set I has bounded cardinality, by which we mean that $|I| \leq \kappa$ for some fixed infinite cardinal κ at least as large as the cardinality of STATES, ACT and PRED. It is also required that $\{A_i\}_{i\in I}$ has finite support; this is then the support of the conjunction. Note that this does not imply that each conjunct has that support, and that we do not require the support of the conjuncts to be uniformly bounded. Alpha-equivalent formulas are identified; the only binding construct is in $\langle\alpha\rangle A$ where $\mathrm{bn}(\alpha)$ binds into A.

The validity of a formula A for a state P is written $P \models A$ and is defined by induction on A as follows.

Definition 5.

$P \models \bigwedge_{i\in I} A_i$	*if for all $i \in I$ it holds that $P \models A_i$*
$P \models \neg A$	*if not $P \models A$*
$P \models \varphi$	*if $P \vdash \varphi$*
$P \models \langle\alpha\rangle A$	*if there exists P' such that $P \xrightarrow{\alpha} P'$ and $P' \models A$*

In the last clause we assume that $\langle\alpha\rangle A$ is a representative of its alpha-equivalence class such that $\mathrm{bn}(\alpha)\#P$. It is easy to show that $\models$ is an equivariant relation.

Definition 6. *Two states P and Q are* logically equivalent*, written $P \dot{=} Q$, if for all A it holds that $P \models A$ iff $Q \models A$*

Theorem 7. $P \dot{\sim} Q \implies P \dot{=} Q$

The proof is by induction over formulas. The converse result uses the idea of *distinguishing formulas*.

Definition 8. *A* distinguishing formula *for P and Q is a formula A such that $P \models A$ and not $Q \models A$.*

Note that if A is a distinguishing formula for P and Q then $\neg A$ is a distinguishing formula for Q and P. Thus P and Q are logically equivalent precisely if there is no distinguishing formula for P and Q. The following lemma says that we can find such a formula where, a bit surprisingly, the support does not depend on Q.

Lemma 9. *If A is a distinguishing formula for P and Q, then there exists a distinguishing formula B for P and Q such that* $\mathrm{supp}(B) \subseteq \mathrm{supp}(P)$.

The proof is by direct construction: in the terminology of Pitts [24, Ch. 5] B is the conjunction of $\mathrm{hull}_{\mathrm{supp}(P)}A$.

Theorem 10. $P \dot{=} Q \implies P \dot{\sim} Q$

Proof: We establish that $\dot{=}$ is a bisimulation. The proof of the simulation property of Theorem 10 is different from earlier similar proofs. These use the name preservation property to show that $\dot{=}$ restricted to states with a finite bound on the names is a bisimulation. This does not hold in the present paper; instead we use Lemma 9 to bound the support of distinguishing formulas.

4 Derived formulas

Dual connectives. We define logical disjunction $\bigvee_{i\in I} A_i$ in the usual way as $\neg \bigwedge_{i\in I} \neg A_i$, when the indexing set I has bounded cardinality and $\{A_i\}_{i\in I}$ has finite support. A special case is $I = \{1,2\}$: we then write $A_1 \wedge A_2$ instead of $\bigwedge_{i\in I} A_i$, and dually for $A_1 \vee A_2$. We write $\top$ for the empty conjunction $\bigwedge_{i\in\emptyset}$, and $\bot$ for $\neg\top$. The must modality $[\alpha]A$ is defined as $\neg\langle\alpha\rangle\neg A$, and requires A to hold after every possible α-labelled transition from the current state. For example, $[\alpha](A \wedge B)$ is equivalent to $[\alpha]A \wedge [\alpha]B$, and dually $\langle\alpha\rangle(A \vee B)$ is equivalent to $\langle\alpha\rangle A \vee \langle\alpha\rangle B$.

Quantifiers. Let S be any finitely supported set of bounded cardinality and use v to range over members of S. Write $A\{^v/_x\}$ for the substitution of v for x in A, and assume this substitution function is equivariant. Then we define $\forall x \in S.A$ as $\bigwedge_{v\in S} A\{^v/_x\}$. There is not necessarily a common finite support for the formulas $A\{^v/_x\}$, for example if S is some term algebra over names, but the set $\{A\{^v/_x\} : v \in S\}$ has finite support bounded by $\{x\} \cup \mathrm{supp}(S) \cup \mathrm{supp}(A)$. In our examples in Section 6, substitution is defined inductively on the structure of formulas, based on primitive substitution functions for actions and state predicates, avoiding capture and preserving the binding names of actions.

Existential quantification $\exists x \in S.A$ is defined as the dual $\neg\forall x \in S.\neg A$. When X is a metavariable used to range over a nominal set $\mathscr{X}$, we simply write X for "$X \in \mathscr{X}$". As an example, $\forall a.A$ means that the formula $A\{^n/_a\}$ holds for all names $n \in \mathscr{N}$.

New name quantifier. The new name quantifier $\mathit{N}x.A$ intuitively states that $P \models A\{^n/_x\}$ holds where n is a fresh name for P. For example, suppose we have actions of the form $a\,b$ for input, and $\overline{a}\,b$ for output where a and b are free names, then the formula $\mathit{N}x.[a\,x]\langle\overline{b}\,x\rangle\top$ expresses that whenever a process inputs a fresh name x on channel a, it has to be able to output that name on channel b. If the name received is not

fresh (i.e., already present in P) then P is not required to do anything. Therefore this formula is weaker than $\forall x.\,[ax]\langle\overline{b}x\rangle\top$.

To define this formally we use name permutation rather than substitution. Since A and P have finite support, if $P \models (xn)\cdot A$ holds for some n fresh for P, by equivariance it also holds for almost all n, i.e., all but finitely many n. Conversely, if it holds for almost all n, it must hold for some $n\#\mathrm{supp}(P)$. Therefore $\mathit{M}x$ is often pronounced "for almost all x". In other words, $P \models \mathit{M}x.A$ holds if $\{x \mid P \models A(x)\}$ is a cofinite set of names [24, Definition 3.8]. Letting $\mathrm{COF} = \{S \subseteq \mathcal{N} \mid \mathcal{N} \setminus S \text{ is finite}\}$ we thus encode $\mathit{M}x.A$ as $\bigvee_{S\in\mathrm{COF}} \bigwedge_{n\in S}(xn)\cdot A$. This formula states there is a cofinite set of names such that for all of them A holds. The support of $\bigwedge_{n\in S}(xn)\cdot A$ is bounded by $(\mathcal{N}\setminus S)\cup\mathrm{supp}(A)$ where $S \in \mathrm{COF}$, and the support of the encoding $\bigvee_{S\in\mathrm{COF}} \bigwedge_{n\in S}(xn)\cdot A$ is bounded by $\mathrm{supp}(A)$.

Next step. We generalise the action modality to sets of actions in the following way. If T is a finitely supported set of actions such that $\mathrm{bn}(\alpha)\#A$ for all $\alpha \in T$, we write $\langle T\rangle A$ for $\bigvee_{\alpha\in T}\langle\alpha\rangle A$. The support of the set $\{\langle\alpha\rangle A : \alpha \in T\}$ is bounded by $\mathrm{supp}(T)\cup\mathrm{supp}(A)$ and thus finite. Dually, we write $[T]A$ for $\neg\langle T\rangle\neg A$, denoting that A holds after all transitions with actions in T.

To encode the next-step modality, we let $\mathrm{ACT}_A = \{\alpha : \mathrm{bn}(\alpha)\#A\}$. Noting that $\mathrm{supp}(\mathrm{ACT}_A) \subseteq \mathrm{supp}(A)$ is finite, we write $\langle\rangle A$ for $\langle\mathrm{ACT}_A\rangle A$, meaning that we can make some (non-capturing) transition to a state where A holds. As an example, $\langle\rangle\top$ means that the current state is not deadlocked. The dual modality $[]A = \neg\langle\rangle\neg A$ means that A holds after every transition from the current state. Larsen [17] uses the same approach to define next-step operators in HML, though his version is less expressive since he uses a finite action set to define the next-step modality.

Fixpoints. Fixpoint operators are a way to introduce recursion into a logic. For example, they can be used to concisely express safety and liveness properties of a transition system, where by safety we mean that some invariant holds for all reachable states, and by liveness that some property will eventually hold. Kozen (1983) [16] introduced the least ($\mu X.A$) and the greatest ($\nu X.A$) fixpoints in modal logic. Intuitively, the least fixpoint states a property that holds for states of a finite path, while the greatest holds for states of an infinite path.

Theorem 11. *The least and greatest fixpoint operators are expressible in our HML.*

For the full proofs and definitions, see the appendix. The idea is to start with an extended language with the forms $\mu X.A$ and X, where X ranges over a countable set of variables and all occurrences of X in A are in the scope of an even number of negations. Write $A(B)$ for the capture-avoiding substitution of B for X in A, and let $A^0(B) = B$ and $A^{i+1}(B) = A(A^i(B))$. Then the encoding of a least fixpoint $\mu X.A$ is $\bigvee_{i\in\mathbb{N}} A^i(\bot)$, given that fixpoints have been recursively expanded in A. The disjunction has finite support $\mathrm{supp}(A)$, since substitution is equivariant. When interpreting formulas as elements of the power-set lattice of STATES, this encoding yields a fixpoint of $A(\cdot)$: the sequence of formulas $A^i(\bot)$ yields an approximation from below. We define the greatest fixpoint operator $\nu X.A$ in terms of the least as $\neg\mu X.\neg A(\neg X)$.

Using the greatest fixpoint operator we can state global invariants: $\nu X.[\alpha]X \wedge A$ expresses that A holds along all paths labelled with α. Temporal operators such as eventually can also be encoded using the least fixpoint operator: the formula $\mu X.\langle\alpha\rangle X \vee A$ states that eventually A holds along some path labelled with α. We can freely mix the fixpoint operators to obtain formulas like $\nu X.[\alpha]X \wedge (\mu Y.\langle\beta\rangle Y \vee A)$ which means that for each state along any path labelled with α, a state where A holds is reachable along a path labelled with β. Formulas with mixed fixpoint combinators are very expressive, and with the next operator they can encode the branching-time logic CTL* [11].

5 Logics for variants of bisimilarity

The bisimilarity of Section 3 is of the early kind: any substitutive effect of an input (typically replacing a variable with the value received) must have manifested already in the action corresponding to the input, since we apply no substitution to the target state. Alternative treatments of substitutions include late-, open- and hyperbisimilarity, where the input action instead contains the variable to be replaced, and there are different ways to make sure that bisimulations are preserved by relevant substitutions.

In our definition of nominal transition systems there are no particular input variables in the states or in the actions, and thus no a priori concept of "substitution". We therefore choose to formulate the alternatives using so called effect functions. An *effect* is simply a finitely supported function from states to states. For example, in the monadic pi-calculus the effects would be the functions replacing one name by another. In a value-passing calculus the effects would be substitutions of values for variables. In the psi-calculi framework the effects would be sequences of parallel substitutions. Our definitions and results are applicable to any of these; our only requirement is that the effects form a nominal set which we designate by $\mathscr{F}$. Variants of bisimilarity then correspond to requiring continuation after various effects. For example, if the action contains an input variable x then the effects appropriate for late bisimilarity would be substitutions for x.

We will formulate these variants as F/L*-bisimilarity*, where F (for *first*) represents the set of effects that must be observed before following a transition, and L (for *later*) is a function that represents how this set F changes depending on the action of a transition, i.e., $L(\alpha,F)$ is the set of effects that must follow the action α if the previous effect set was F. In the following let $\mathscr{P}_{\mathrm{fs}}(\mathscr{F})$ ranged over by F be the finitely supported subsets of $\mathscr{F}$, and L range over equivariant functions from actions and $\mathscr{P}_{\mathrm{fs}}(\mathscr{F})$ to $\mathscr{P}_{\mathrm{fs}}(\mathscr{F})$.

Definition 12. *An* L-bisimulation *where* $L : \mathrm{ACT} \times \mathscr{P}_{fs}(\mathscr{F}) \to \mathscr{P}_{fs}(\mathscr{F})$ *is a* $\mathscr{P}_{fs}(\mathscr{F})$*-indexed family of symmetric binary relations on states satisfying the following:*

If $R_F(P,Q)$ *then:*

1. Static implication*: for all* $f \in F$ *it holds that* $f(P) \vdash \varphi$ *implies* $f(Q) \vdash \varphi$.
2. Simulation*: For all* $f \in F$ *and* α, P' *such that* $\mathrm{bn}(\alpha)\#f(Q)$ *there exist* Q' *such that*

$$\text{if } f(P) \xrightarrow{\alpha} P' \text{ then } f(Q) \xrightarrow{\alpha} Q' \text{ and } R_{L(\alpha,F)}(P',Q')$$

We write $P \overset{F/L}{\sim} Q$*, called* F/L*-bisimilarity, to mean that there exists an L-bisimulation R such that* $R_F(P,Q)$.

Most strong bisimulation varieties can be formulated as F/L-bismilarity. Write $\mathrm{id}_{\mathrm{STATES}}$ for the identity function on states, ID for the singleton set $\{\mathrm{id}_{\mathrm{STATES}}\}$ and $\mathrm{all}_{\mathrm{ID}}$ for the constant function $\lambda(\alpha,F).\mathrm{ID}$.

- *Early bisimilarity*, precisely as defined in Definition 2, is $\mathrm{ID}\,/\,\mathrm{all}_{\mathrm{ID}}$-bisimilarity.
- *Early equivalence*, i.e. early bisimilarity under all possible effects, is $\mathscr{F}\,/\,\mathrm{all}_{\mathrm{ID}}$-bisimilarity.
- *Late bisimilarity* is $\mathrm{ID}\,/\,L$-bisimilarity, where $L(\alpha,F)$ yields the effects that represent substitutions for variables in input actions α (and ID for other actions).
- *Late equivalence* is similarly $\mathscr{F}\,/\,L$-bisimilarity.
- *Open bisimilarity* is $\mathscr{F}\,/\,L$-bisimilarity where $L(\alpha,F)$ is the set F minus all effects that change bound output names in α.

- *Hyperbisimilarity* is $\mathscr{F}/\lambda(\alpha,F).\mathscr{F}$-bisimilarity.

All of the above are generalizations of known and well-studied definitions. The original value-passing variant of CCS [18] uses early bisimilarity. The original bisimilarity for the pi-calculus is of the late kind [19], where it also was noted that late equivalence is the corresponding congruence. Early bisimilarity and equivalence and open bisimilarity for the pi-calculus were introduced in 1993 [20, 25], and hyperbisimilarity for the fusion calculus in 1998 [22].

In view of this we only need to provide a modal logic adequate for F/L-bisimilarity; it can then immediately be specialized to all of the above variants. For this we introduce a new kind of logical operator as follows.

Definition 13. *For each $f \in \mathscr{F}$ the logical unary* effect consequence *operator $\langle f\rangle$ has the definition*

$$P \models \langle f\rangle A \quad \textit{if} \quad f(P) \models A$$

Thus the formula $\langle f\rangle A$ means that A holds if the effect f is applied to the state. Note that by definition this distributes over conjunction and negation, e.g. $P \models \neg\langle f\rangle A$ iff $P \models \langle f\rangle\neg A$ iff not $f(P) \models A$ etc. The effect consequence operator is similar in spirit to the action modalities: both $\langle f\rangle A$ and $\langle\alpha\rangle A$ assert that something (an effect or action) must be possible and that A holds afterwards. Indeed, effects can be viewed as a special case of transitions (as formalised in Definition 17 below) which is why we give the operators a common syntactic appearance.

Now define the formulas that can directly use effects from F and after actions use effects according to L, ranged over by $A^{F/L}$, in the following way:

Definition 14. *Given L as in Definition 12, for all $F \in \mathscr{P}_{fs}(\mathscr{F})$ define $\mathscr{A}^{F/L}$ as the set of formulas given by the mutually recursive definitions:*

$$A^{F/L} \quad ::= \quad \bigwedge_{i\in I} A_i^{F/L} \quad | \quad \neg A^{F/L} \quad | \quad \langle f\rangle\varphi \quad | \quad \langle f\rangle\langle\alpha\rangle A^{L(\alpha,F)/L}$$

where we require $f \in F$ and that the conjunction has bounded cardinality and finite support.

Let $P \overset{F/L}{=} Q$ mean that P and Q satisfy the same formulas in $\mathscr{A}^{F/L}$.

Theorem 15. $P \overset{F/L}{\sim} Q \quad \Leftrightarrow \quad P \overset{F/L}{=} Q$

Proof: The direction $\Rightarrow$ is a generalization of Theorem 7. The other direction is a generalization of Theorem 10: we prove that $\overset{F/L}{=}$ is an F/L-bisimulation. It needs a variant of Lemma 9:

Lemma 16. *If $A \in \mathscr{A}^{F/L}$ is a distinguishing formula for P and Q, then there exists a distinguishing formula $B \in \mathscr{A}^{F/L}$ for P and Q such that* $\mathrm{supp}(B) \subseteq \mathrm{supp}(P,F)$.

The proof is an easy generalisation of Lemma 9.

An alternative to the effect consequence operators is to transform the transition system such that standard (early) bisimulation on the transforms coincides with F/L-bisimilarity. The idea is to let the effect function be part of the transition relation, thus $f(P) = P'$ becomes $P \xrightarrow{f} P'$.

Definition 17. *Assume $\mathscr{F}$ and L as above. The L-*transform *of a nominal transition system* $\mathbf{T}$ *is a nominal transition system where:*

- *The states are of the form* $\mathrm{AC}(F, f(P))$ *and* $\mathrm{EF}(F,P)$*, for $f \in F \in \mathscr{P}_{fs}(\mathscr{F})$ and states P of* $\mathbf{T}$*. The intuition is that states of kind* AC *can perform ordinary actions, and states of kind* EF *can commit effects.*

- *The state predicates are those of* $\mathbf{T}$.
- $\mathrm{AC}(F,P) \vdash \varphi$ *if in* $\mathbf{T}$ *it holds* $P \vdash \varphi$, *and* $\mathrm{EF}(F,P) \vdash \varphi$ *never holds.*
- *The actions are the actions of* $\mathbf{T}$ *and the effects in* $\mathscr{F}$.
- bn *is as in* $\mathbf{T}$, *and additionally* $\mathrm{bn}(f) = \emptyset$ *for* $f \in \mathscr{F}$.
- *The transitions are of two kinds. If in* $\mathbf{T}$ *it holds* $P \xrightarrow{\alpha} P'$, *then there is a transition* $\mathrm{AC}(F,P) \xrightarrow{\alpha} \mathrm{EF}(L(\alpha,F),P')$. *And for each* $f \in F$ *it holds* $\mathrm{EF}(F,P) \xrightarrow{f} \mathrm{AC}(F,f(P))$.

Theorem 18. $P \overset{F/L}{\sim} Q$ *in* $\mathbf{T}$ *if and only if* $\mathrm{EF}(F,P) \dot{\sim} \mathrm{EF}(F,Q)$ *in the L-transform of* $\mathbf{T}$.

The proof idea is that from an F/L-bisimulation in $\mathbf{T}$ it is easy to construct an (ordinary) bisimulation in the L-transform of $\mathbf{T}$, and vice versa. A direct consequence is that $P \overset{F/L}{\sim} Q$ iff $\mathrm{EF}(F,P) \doteq \mathrm{EF}(F,Q)$ in the L-transform of $\mathbf{T}$. Here the actions in the logic would include effects $f \in \mathscr{F}$.

6 Related work and examples

HML for CCS. The first published HML is Hennessy and Milner (1985) [15]. They use finite (binary) conjunction with the assumption of image-finiteness for ordinary CCS. The same goes for the value-passing calculus and logic by Hennessy and Liu (1995) [14], where image-finiteness is due to a late semantics and the logic contains quantification over data values. A similar idea and argument is in a logic for LOTOS by Calder et al. (2002) [8], though that only considers stratified bisimilarity up to ω.

Hennessy and Liu's value-passing calculus is based on abstractions $(x)P$ and concretions (v,P) where v is drawn from a set of values. To encode their logic in ours, we add effects $\mathrm{id}_{\mathrm{STATES}}$ and $?v$, with $?v((x)P) = P\{^v/_x\}$, and transitions $(v,P) \xrightarrow{!v} P$. Letting $L(a?,\_) = \{?v \ : \ v \in \text{values}\}$ and $L(\alpha,\_) = \{\mathrm{id}_{\mathrm{STATES}}\}$ otherwise, late bisimilarity is $\{\mathrm{id}_{\mathrm{STATES}}\}/L$-bisimilarity as defined in Section 5. We can then encode their universal quantifier $\forall x.A$ as $\bigwedge_v \langle ?v\rangle A\{^v/_x\}$, which has support $\mathrm{supp}(A) \setminus \{x\}$, and their output modality $\langle c!x\rangle A$ as $\langle c!\rangle \bigvee_v \langle !v\rangle A\{^v/_x\}$, with support $\{c\} \cup (\mathrm{supp}(A) \setminus \{x\})$.

An infinitary HML for CCS is discussed in Milner's book (1989) [18], where also the process syntax contains infinite summation. There are no restrictions on the indexing sets and no discussion about how this can exhaust all names. The adequacy theorem is proved by stratifying bisimilarity and using transfinite induction over all ordinals, where the successor step basically is the contraposition of the argument in Theorem 10, though without any consideration of finite support. A more rigorous treatment of the same ideas is by Abramsky (1991) [3] where uniformly bounded conjunction is used throughout.

Pi-calculus. The first HML for the pi-calculus is by Milner et al. (1993) [20], where infinite conjunction is used in the early semantics and conjunctions are restricted to use a finite set of free names. The adequacy proof is of the same structure as in this paper. The logic defined in this paper, applied to the pi-calculus transition system omitting bound input actions $x(y)$, contains the logic $\mathscr{F}$ of Milner et al., or the equipotent logic $\mathscr{F}\mathscr{M}$ if we take the set of name matchings $[a=b]$ as state predicates.

Spi Calculus. Frendrup et al. (2002) [12] provide three Hennessy-Milner logics for the spi calculus [2]. The action modalities in Frendrup's logic only uses parts of the labels: on process output, the modality $\langle \overline{a}\rangle$ tests only the channel used. On process input, the modality $\langle a\xi\rangle$ describes how the observer σ computed the received message $M = \mathbf{e}(\xi\sigma)$, where ξ is an expression that may contain decryptions and projections, and $\mathrm{supp}(\xi) \setminus \mathrm{dom}(\sigma)$ is fresh for P and σ. Simplifying the labels of the transition

system to τ and the aforementioned $\overline{a}$ and $a\xi$ labels, our minimal HML applied to the particular nominal transition system of the spi calculus coincides with the logic $\mathscr{F}$ of Frendrup et al, although the latter uses infinite conjunction without any mechanism to prevent formulas from exhausting all names, leaving none available for alpha-conversion. Thus their notion of substitution is not formally well defined.

Their logic $\mathscr{EM}$ replaces the simple input modality by an early input modality $\langle \underline{a}(x) \rangle^E A$, which (after a minor manipulation of the input labels) can be encoded as the conjunction $\bigwedge_\xi \langle a\xi \rangle A\{^\xi/_x\}$, which has support $\mathrm{supp}(A) \setminus \{x\}$. We do not consider their logic $\mathscr{LM}$ that uses a late input modality, since its application relies on sets that do not have finite support [12, Theorem 6.12], which are not meaningful in nominal logic.

Applied Pi-calculus. A more recent work is a logic by Pedersen (2006) [23] for the applied pi-calculus [1], where the adequacy theorem uses image-finiteness of the semantics in the contradiction argument. The logic contains atomic formulae for equality in the frame of a process, corresponding to our state predicates. The main difference to our logic is an early input modality and a quantifier $\exists x$.

Their early input modality $\langle \underline{a}(x) \rangle A$ can be straightforwardly encoded as the conjunction $\bigwedge_M \langle \underline{a}M \rangle A\{^M/_x\}$, with support $\{a\} \cup (\mathrm{supp}(A) \setminus \{x\})$. For the existential quantifier, there is a requirement that the received term M can be computed from the current knowledge available to an observer of the process, which we here write $M \in \mathscr{S}(P)$. We add actions M/x with $\mathrm{bn}(M/x) = x$ and transitions $P \xrightarrow{M/x} P \mid \{^M/_x\}$ if $M \in \mathscr{S}(P)$ and $x\#P$. We can then encode $\exists x.A$ as $\bigvee_M \langle M/x \rangle A$, which has support $\mathrm{supp}(A) \setminus \{x\}$.

Fusion calculus. In a HML for the fusion calculus by Haugstad et al. (2006) [13] the fusions (i.e., equality relations on names) are action labels φ. The corresponding modal operator $\langle \varphi \rangle A$ has the semantics that the formula A must be satisfied for all substitutive effects of φ (intuitively, substitutions that map each name to a fixed representative for its equivalence class). The adequacy theorem uses the contradiction argument with infinite conjunction, with no argument about finiteness of names for the distinguishing formula. This HML can be encoded in our framework by making the substitutive effects of fusion actions visible in the transition system.

Concurrent constraint pi calculus. The concurrent constraint pi calculus (CC-pi) by Buscemi and Montanari (2007) [6] extends the explicit fusion calculus [27] with a more general notion of constraint stores c. The reference equivalence for CC-pi is open bisimulation [7] (closely corresponding to hyperbisimulation in the fusion calculus [22]), which differs from labelled bisimulation in two ways: First, two equivalent processes must be equivalent under all store extensions. To encode this, we let the effects $\mathscr{F}$ be the set of constraint stores c different from 0, and let $c(P) = c \mid P$. Second, when simulating a labelled transition $P \xrightarrow{\alpha} P'$, the simulating process Q can use any transition $Q \xrightarrow{\beta} Q'$ with an equivalent label, as given by a state predicate $\alpha = \beta$. As an example, if $\alpha = \overline{a}\langle x \rangle$ is a free output label then $P \vdash \alpha = \beta$ iff $\beta = \overline{b}\langle y \rangle$ where $P \vdash a = b$ and $P \vdash x = y$. To encode this, we transform the labels of the transition system by replacing them with their equivalence classes, i.e., $P \xrightarrow{\alpha} P'$ becomes $P \xrightarrow{[\alpha]_P} P'$ where $\beta \in [\alpha]_P$ iff $P \vdash \beta = \alpha$. Hyperbisimilarity (Definition 12) on this transition system then corresponds to open bisimilarity, and the modal logic defined in Section 5 is adequate.

Nominal transition systems. De Nicola and Loreti (2008) [10] define a general format for nominal transition systems and an associated modal logic, that is adequate for image-finite transition systems only and uses several different modalities for name revelation and resource consumption. In contrast, we seek a small and expressive HML for general nominal transition systems. Indeed, the logic of De Nicola

and Loreto can be seen as a special case of ours: their different transition systems can be merged into a single one, and we can encode their quantifiers and fixpoint operator as described in Section 4. Nominal SOS of Cimini et al. (2012) [9] is also a special case of nominal transition systems.

Psi-calculi. In psi-calculi by Bengtson et al (2011) [4], the labelled transitions take the form $\Psi \triangleright P \xrightarrow{\alpha} P'$, where the assertion environment Ψ is unchanged after the step. We model this as a nominal transition system by letting the set of states be pairs (Ψ, P) of assertion environments and processes, and define the transition relation by $(\Psi, P) \xrightarrow{\alpha} (\Psi, P')$ if $\Psi \triangleright P \xrightarrow{\alpha} P'$. The notion of bisimulation used with psi-calculi also uses an assertion environment and is required to be closed under environment extension, i.e., if $\Psi \triangleright P \sim Q$, then $\Psi \otimes \Psi' \triangleright P \sim Q$ for all Ψ'. We let the effects $\mathscr{F}$ be the set of assertions, and define $\Psi((\Psi', P)) = (\Psi \otimes \Psi', P)$. Hyperbisimilarity on this transition system then subsumes the standard psi-calculi bisimilarity, and the modal logic defined in Section 5 is adequate.

7 Conclusion

We have given a general account of transition systems and Hennessy-Milner Logic using nominal sets. The advantage of our approach is that it is more expressive than previous work. We allow infinite conjunctions that are not uniformly bounded, meaning that we can encode e.g. quantifiers and the next-step operator. We have given ample examples of how the definition captures different variants of bisimilarity and how it relates to many different versions of HML in the literature.

We have formalized the results of Section 3, including Theorems 7 and 10, using Nominal Isabelle [26].[1] Nominal Isabelle is an implementation of nominal logic in Isabelle/HOL [21], a popular interactive proof assistant for higher-order logic. It adds convenient specification mechanisms for, and automation to reason about, datatypes with binders.

However, Nominal Isabelle does not directly support infinitely branching datatypes. Therefore, the mechanization of formulas (Definition 4) was challenging. We construct formulas from first principles in higher-order logic, by defining an inductive datatype of *raw* formulas (where alpha-equivalent raw formulas are *not* identified). The datatype constructor for conjunction recurses through sets of raw formulas of bounded cardinality, a feature made possible only by a recent re-implementation of Isabelle/HOL's datatype package [5].

We then define alpha-equivalence of raw formulas. For finitely branching datatypes, alpha-equivalence is based on a notion of free variables. Here, to obtain the correct notion of free variables of a conjunction, we define alpha-equivalence and free variables via mutual recursion. This necessitates a fairly involved termination proof. (All recursive functions in Isabelle/HOL must be terminating.) To obtain formulas, we quotient raw formulas by alpha-equivalence, and finally carve out the subtype of all terms that can be constructed from finitely supported ones. We then prove important lemmas; for instance, a strong induction principle for formulas that allows the bound names in $\langle\alpha\rangle A$ to be chosen fresh for any finitely supported context.

Our development, which in total consists of about 2700 lines of Isabelle definitions and proofs, generalizes the constructions that Nominal Isabelle performs for finitely branching datatypes to a type with infinite branching. To our knowledge, this is the first mechanization of an infinitely branching nominal datatype in a proof assistant.

[1] Our Isabelle theories are available at `https://github.com/tjark/ML-for-NTS`.

Acknowledgements

We thank Andrew Pitts for enlightening discussions on nominal datatypes with infinitary constructors, and Dmitriy Traytel for providing a formalization of cardinality-bounded sets.

References

[1] Martín Abadi & Cédric Fournet (2001): *Mobile Values, New Names, and Secure Communication*. In: *Proceedings of POPL '01*, ACM, pp. 104–115.

[2] Martín Abadi & Andrew D. Gordon (1999): *A Calculus for Cryptographic Protocols: The Spi Calculus*. *Journal of Information and Computation* 148(1), pp. 1–70.

[3] Samson Abramsky (1991): *A Domain Equation for Bisimulation*. *Journal of Information and Computation* 92(2), pp. 161–218, doi:10.1006/inco.1991.9999. Available at `http://dx.doi.org/10.1006/inco.1991.9999`.

[4] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1), doi:10.2168/LMCS-7(1:11)2011. Available at `http://dx.doi.org/10.2168/LMCS-7(1:11)2011`.

[5] Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu & Dmitriy Traytel (2014): *Truly Modular (Co)datatypes for Isabelle/HOL*. In Gerwin Klein & Ruben Gamboa, editors: *Proceedings of ITP 2014*, *LNCS* 8558, Springer, pp. 93–110, doi:10.1007/978-3-319-08970-6_7. Available at `http://dx.doi.org/10.1007/978-3-319-08970-6_7`.

[6] Maria Grazia Buscemi & Ugo Montanari (2007): *CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements*. In Rocco De Nicola, editor: *Proceedings of ESOP 2007*, *LNCS* 4421, Springer, pp. 18–32.

[7] Maria Grazia Buscemi & Ugo Montanari (2008): *Open Bisimulation for the Concurrent Constraint Pi-Calculus*. In Sophia Drossopoulou, editor: *Proceedings of ESOP 2008*, *LNCS* 4960, Springer, pp. 254–268, doi:10.1007/978-3-540-78739-6_20. Available at `http://dx.doi.org/10.1007/978-3-540-78739-6_20`.

[8] Muffy Calder, Savi Maharaj & Carron Shankland (2002): *A modal logic for full LOTOS based on symbolic transition systems*. *The Computer Journal* 45(1), pp. 55–61.

[9] Matteo Cimini, Mohammad Reza Mousavi, Michel A. Reniers & Murdoch J. Gabbay (2012): *Nominal SOS*. *Electron. Notes Theor. Comput. Sci.* 286, pp. 103–116, doi:10.1016/j.entcs.2012.08.008. Available at `http://dx.doi.org/10.1016/j.entcs.2012.08.008`.

[10] Rocco De Nicola & Michele Loreti (2008): *Multiple-Labelled Transition Systems for nominal calculi and their logics*. *Mathematical Structures in Computer Science* 18(1), pp. 107–143, doi:10.1017/S0960129507006585. Available at `http://dx.doi.org/10.1017/S0960129507006585`.

[11] E. Allen Emerson (1997): *Model checking and the Mu-calculus*. In: *DIMACS Series in Discrete Mathematics*, American Mathematical Society, pp. 185–214.

[12] Ulrik Frendrup, Hans Hüttel & Jesper Nyholm Jensen (2002): *Modal Logics for Cryptographic Processes*. *Electr. Notes Theor. Comput. Sci.* 68(2), pp. 124–141. Available at `http://dx.doi.org/10.1016/S1571-0661(05)80368-8`.

[13] Arild Martin Møller Haugstad, Anders Franz Terkelsen & Thomas Vindum (2006): *A Modal Logic for the Fusion Calculus*. Unpublished, University of Aalborg.

[14] M. Hennessy & X. Liu (1995): *A modal logic for message passing processes*. *Acta Informatica* 32(4), pp. 375–393, doi:10.1007/BF01178384. Available at `http://dx.doi.org/10.1007/BF01178384`.

[15] Matthew Hennessy & Robin Milner (1985): *Algebraic Laws for Nondeterminism and Concurrency*. *J. ACM* 32(1), pp. 137–161. Available at `http://doi.acm.org/10.1145/2455.2460`.

[16] Dexter Kozen (1983): *Results on the propositional mu-calculus*. Theoretical Computer Science 27(3), pp. 333 – 354, doi:http://dx.doi.org/10.1016/0304-3975(82)90125-6. Available at http://www.sciencedirect.com/science/article/pii/0304397582901256. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[17] Kim G. Larsen (1988): *Proof systems for Hennessy-Milner Logic with recursion*. In M. Dauchet & M. Nivat, editors: *Proceedings of CAAP '88*, *LNCS* 299, Springer, pp. 215–230, doi:10.1007/BFb0026106. Available at http://dx.doi.org/10.1007/BFb0026106.

[18] Robin Milner (1989): *Communication and Concurrency*. Prentice Hall.

[19] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I*. *Inf. Comput.* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4. Available at http://dx.doi.org/10.1016/0890-5401(92)90008-4.

[20] Robin Milner, Joachim Parrow & David Walker (1993): *Modal logics for mobile processes*. Theoretical Computer Science 114(1), pp. 149 – 171, doi:10.1016/0304-3975(93)90156-N. Available at http://www.sciencedirect.com/science/article/pii/030439759390156N.

[21] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. *LNCS* 2283, Springer, doi:10.1007/3-540-45949-9. Available at http://dx.doi.org/10.1007/3-540-45949-9.

[22] Joachim Parrow & Björn Victor (1998): *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*. In: *Proceedings of LICS 1998*, pp. 176–185, doi:10.1109/LICS.1998.705654. Available at http://dx.doi.org/10.1109/LICS.1998.705654.

[23] Michael Pedersen (2006): *Logics for the Applied pi calculus*. Master's thesis, Aalborg University. BRICS RS-06-19.

[24] Andrew M. Pitts (2013): *Nominal Sets*. Cambridge University Press. Available at http://dx.doi.org/10.1017/CBO9781139084673. Cambridge Books Online.

[25] Davide Sangiorgi (1993): *A theory of bisimulation for the π-calculus*. In Eike Best, editor: *Proceedings of CONCUR '93*, *LNCS* 715, Springer, pp. 127–142, doi:10.1007/3-540-57208-2_10. Available at http://dx.doi.org/10.1007/3-540-57208-2_10.

[26] Christian Urban & Cezary Kaliszyk (2012): *General Bindings and Alpha-Equivalence in Nominal Isabelle*. Logical Methods in Computer Science 8(2), doi:10.2168/LMCS-8(2:14)2012. Available at http://dx.doi.org/10.2168/LMCS-8(2:14)2012.

[27] Lucian Wischik & Philippa Gardner (2005): *Explicit fusions*. Theoretical Computer Science 304(3), pp. 606–630.

A Appendix: Proofs

Proofs from Section 3

Proposition 3 $\dot{\sim}$ is an equivariant equivalence relation.

Proof: The proof has been formalized in Isabelle. Equivariance is a simple calculation, based on the observation that if R is a bisimulation, then $\pi \cdot R$ is a bisimulation. To prove reflexivity of $\dot{\sim}$, we note that equality is a bisimulation. Symmetry is immediate from Def. 2. To prove transitivity, we show that the composition of $\dot{\sim}$ with itself is a bisimulation; the simulation requirement is proved by a considering an alpha-variant of $P \xrightarrow{\alpha} P'$ where $\mathrm{bn}(\alpha)$ is fresh for Q. □

Lemma 19. $\models$ *is equivariant.*

Proof: By the Equivariance Principle in Pitts (2013) [24, page 21]. A more detailed proof that verifies $P \models A \iff \pi \cdot P \models \pi \cdot A$ for any permutation π has been formalized in Isabelle. The proof proceeds

by structural induction on A, using equivariance of all involved relations. For the case $\langle\alpha\rangle A$ in particular, we use the fact that if $\langle\alpha'\rangle A' = \langle\alpha\rangle A$, then $\langle\pi\cdot\alpha'\rangle(\pi\cdot A') = \langle\pi\cdot\alpha\rangle(\pi\cdot A)$. □

Theorem 7 $P \dot{\sim} Q \implies P \dot{=} Q$

Proof: The proof has been formalized in Isabelle. Assume $P \dot{\sim} Q$. We show $P \models A \iff Q \models A$ by structural induction on A.

1. Base case: $A = \varphi$. Then $P \models A \iff P \vdash \varphi \iff Q \vdash \varphi \iff Q \models A$ by static implication and symmetry of $\dot{\sim}$.
2. Inductive steps $\bigwedge_{i\in I} A_i$ and $\neg A$: immediate by induction.
3. Inductive step $\langle\alpha\rangle A$: Assume $P \models \langle\alpha\rangle A$. Then for some alpha-variant $\langle\alpha'\rangle A' = \langle\alpha\rangle A$, $\exists P'.\, P \xrightarrow{\alpha'} P'$ and $P' \models A'$. Without loss of generality we assume also $\mathrm{bn}(\alpha')\#Q$, otherwise just find an alpha-variant of $\langle\alpha'\rangle A'$ where this holds. Then by simulation $\exists Q'.\, Q \xrightarrow{\alpha'} Q'$ and $P' \dot{\sim} Q'$. By induction and $P' \models A'$ we get $Q' \models A'$, whence by definition $Q \models \langle\alpha\rangle A$. The proof of $Q \models \langle\alpha\rangle A \implies P \models \langle\alpha\rangle A$ is symmetric, using the fact that $P \dot{\sim} Q$ entails $Q \dot{\sim} P$. □

Lemma 9 If A is a distinguishing formula for P and Q, then there exists a distinguishing formula B for P and Q such that $\mathrm{supp}(B) \subseteq \mathrm{supp}(P)$.

Proof: The proof has been formalized in Isabelle. We use the fact that a conjunction is well-formed if the set of conjuncts has finite support—this is much more liberal than that each conjunct has the same finite support. Assume $P \models A$ and not $Q \models A$. Let S be the group of name permutations of names outside $\mathrm{supp}(P)$ and let $\mathscr{B}$ be the S-orbit of A, i.e.,

$$\mathscr{B} = \{\pi\cdot A \mid \pi \in S\}$$

Clearly $\pi'\cdot\mathscr{B} = \mathscr{B}$ for all $\pi' \in S$, thus $\mathrm{supp}(\mathscr{B}) \subseteq \mathrm{supp}(P)$, which means that the formula $B = \bigwedge\mathscr{B}$ is well-formed and $\mathrm{supp}(B) \subseteq \mathrm{supp}(P)$. By equivariance, if $P \models A$ and $\pi \in S$ then also $P \models \pi\cdot A$. Therefore $P \models B$. Furthermore, since the identity permutation is in S and not $Q \models A$ we get not $Q \models B$. □

Theorem 10 $P \dot{=} Q \implies P \dot{\sim} Q$

Proof: The proof has been formalized in Isabelle. We establish that $\dot{=}$ is a bisimulation. Obviously it is symmetric. So assume $P \dot{=} Q$, we need to prove the two requirements on a bisimulation.

1. Static implication. $P \vdash \varphi$ iff $P \models \varphi$ iff $Q \models \varphi$ iff $Q \vdash \varphi$.
2. Simulation. The proof is by contradiction. Assume that $\dot{=}$ does not satisfy the simulation requirement. Then there exist P, Q, P', α with $\mathrm{bn}(\alpha)\#Q$ such that $P \dot{=} Q$ and $P \xrightarrow{\alpha} P'$ and, letting $\mathscr{Q} = \{Q' \mid Q \xrightarrow{\alpha} Q'\}$, for all $Q' \in \mathscr{Q}$ it holds that $P' \dot{\neq} Q'$. Assume $\mathrm{bn}(\alpha)\#P$, othertwise just find an alpha-variant of the transition satisfying this. By $P' \dot{\neq} Q'$, for all $Q' \in \mathscr{Q}$ there exists a distinguishing formula for P' and Q'. The formula may depend on Q', and by Lemma 9 we can find such a distinguishing formula $B_{Q'}$ for P' and Q' with $\mathrm{supp}(B_{Q'}) \subseteq \mathrm{supp}(P')$. Therefore the formula

$$B = \bigwedge_{Q'\in\mathscr{Q}} B_{Q'}$$

is well-formed with support included in $\mathrm{supp}(P')$. We thus get that $P \models \langle\alpha\rangle B$ but not $Q \models \langle\alpha\rangle B$, contradicting $P \dot{=} Q$. □

Proofs from Section 4 on fixpoint operators

Definition 20. *We extend the nominal set of formulas with the least fixpoint operator:*

$$A \quad ::= \quad \bigwedge_{i\in I} A_i \quad | \quad \neg A \quad | \quad \varphi \quad | \quad \langle\alpha\rangle A \quad | \quad X \quad | \quad \mu X.A$$

where X is ranges over a countable set of equivariant variables. We require that all occurrences of a variable X in a formula $\mu X.A$ are in the scope of an even number of negations.

We use a capture-avoiding substitution function on formulas $[A/X]$ that substitutes A for the variable X. In particular, $(\langle\alpha\rangle B)[A/X] = \langle\alpha\rangle(B[A/X])$ when $\mathrm{bn}(\alpha)$ is fresh for A.

To encode the fixpoint operators, we first give a semantics to formulas (including fixpoints) as sets of states.

Definition 21. *A valuation function ε is a map from variables to finitely supported sets of states, such that $\varepsilon(X)$ is the empty set for all but finitely many X. We define the interpretation function as follows:*

$$\begin{array}{rcl}
[\![\bigwedge_{i\in I} A_i]\!]_\varepsilon & = & \bigcap_{i\in I} [\![A_i]\!]_\varepsilon \\
[\![\neg A]\!]_\varepsilon & = & \mathrm{STATES} - [\![A]\!]_\varepsilon \\
[\![\varphi]\!]_\varepsilon & = & \{P \mid P \vdash \varphi\} \\
[\![\langle\alpha\rangle A]\!]_\varepsilon & = & \left\{P \mid \exists P'.\, P \xrightarrow{\alpha} P' \text{ and } P' \in [\![A]\!]_\varepsilon\right\} \\
[\![X]\!]_\varepsilon & = & \varepsilon(X) \\
[\![\mu X.A]\!]_\varepsilon & = & \bigcap\left\{S \in \mathscr{P}_{fs}(\mathrm{STATES}) \mid [\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq S\right\}
\end{array}$$

Note that $[\![\cdot]\!]_\cdot$ is equivariant, so the intersections defining $[\![\bigwedge_{i\in I} A_i]\!]_\varepsilon$ and $[\![\mu X.A]\!]_\varepsilon$ are finitely supported.

Lemma 22. *Let A be a formula as defined in Definition 4, then $P \models A$ if and only if $P \in [\![A]\!]_\varepsilon$.*

Proof. By induction on A. □

Similarly to the Knaster-Tarksi fixpoint theorem, the least fixpoint $\mathrm{lfp}(F)$ of a finitely supported function F on the lattice of finitely supported subsets of a nominal set X can be computed by taking the intersection of all pre-fixpoints (i.e., finitely supported sets S such that $F(S) \subseteq S$). (Note that the usual Tarski fixpoint theorem does not apply, since this lattice is not continuous in general.)

Lemma 23. *If X is a nominal set and $F : \mathscr{P}_{fs}(X) \to \mathscr{P}_{fs}(X)$ is monotonic and finitely supported, then the least fixpoint of F is given by $\bigcap\left\{S \in \mathscr{P}_{fs}(X) \mid F(S) \subseteq S\right\}$.*

Proof. Let $A = \left\{S \in \mathscr{P}_{\mathrm{fs}}(X) \mid F(S) \subseteq S\right\}$. Note that $X \in A$, that A has finite support (bounded by $\mathrm{supp}(F)$), and that A contains all fixpoints of F. Let $C = \bigcap A$; note that $\mathrm{supp}(C) \subseteq \mathrm{supp}(A)$ is finite and that C is a subset of any fixpoint of F. Since $C \subseteq S$ for all $S \in A$, we have $F(C) \subseteq \bigcap_{S\in A} F(S) \subseteq \bigcap_{S\in A} S = C$. From $F(C) \subseteq C$ and monotonicity we get $F(F(C)) \subseteq F(C)$, so $F(C) \in A$ and we get $C \subseteq F(C)$ by construction. From $F(C) \subseteq C \subseteq F(C)$ we get $C = F(C)$, so C is indeed the least fixpoint of F. □

We thus only need to show that $F(S) = [\![A]\!]_{\varepsilon[X\mapsto S]}$ is monotonic in order for the interpretation of the least fixpoint formula $\mu X.A$ to indeed denote the least fixpoint.

Lemma 24. *The function $F(S) = [\![A]\!]_{\varepsilon[X\mapsto S]}$ is monotonic for a closed formula $\mu X.A$.*

Proof. Assume $S \subseteq T$ for finitely supported sets of states S and T. We need to show that $[\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq [\![A]\!]_{\varepsilon[X\mapsto T]}$. We proceed by structural induction on A.

Case φ: Since X does not occur in φ, $[\![\varphi]\!]_{\varepsilon[X\mapsto S]} = [\![\varphi]\!]_{\varepsilon} \subseteq [\![\varphi]\!]_{\varepsilon} = [\![\varphi]\!]_{\varepsilon[X\mapsto T]}$.

Case X': If $X = X'$, then $\varepsilon[X \mapsto S](X) = S \subseteq T = \varepsilon[X \mapsto T](X)$ by assumption. Otherwise $\varepsilon(X) = \varepsilon(X)$ by reflexivity of $\subseteq$.

Case $\neg A$: As induction hypothesis we have $[\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq [\![A]\!]_{\varepsilon[X\mapsto T]}$. We consider two cases. If X does not occur in A, then $[\![\neg A]\!]_{\varepsilon[X\mapsto S]} = [\![\neg A]\!]_{\varepsilon}$ and $[\![\neg A]\!]_{\varepsilon[X\mapsto T]} = [\![\neg A]\!]_{\varepsilon}$, and by reflexivity of $\subseteq$ we are done in this case. Otherwise, the X occurs in A and is in scope of one $\neg$, we know $[\![A]\!]_{\varepsilon[X\mapsto T]} \subseteq [\![A]\!]_{\varepsilon[X\mapsto S]}$, therefore $[\![\neg A]\!]_{\varepsilon[X\mapsto S]} = \text{STATES} - [\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq \text{STATES} - [\![A]\!]_{\varepsilon[X\mapsto T]} = [\![\neg A]\!]_{\varepsilon[X\mapsto S]}$.

Case $\langle\alpha\rangle A$: Since, from induction hypothesis, $[\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq [\![A]\!]_{\varepsilon[X\mapsto T]}$, it is easy to see that $[\![\langle\alpha\rangle A]\!]_{\varepsilon[X\mapsto S]}$ contains at least the state of $[\![\langle\alpha\rangle A]\!]_{\varepsilon[X\mapsto T]}$, that is, the former is the subset of latter.

Case $\bigwedge_{i\in I} A_i$: By induction hypothesis, for every $i \in I$, $[\![A_i]\!]_{\varepsilon[X\mapsto S]} \subseteq [\![A_i]\!]_{\varepsilon[X\mapsto S]}$. Thus, $[\![\bigwedge_{i\in I} A_i]\!]_{\varepsilon[X\mapsto S]} = \bigcap_{i\in I}[\![A_i]\!]_{\varepsilon[X\mapsto S]} \subseteq \bigcap_{i\in I}[\![A_i]\!]_{\varepsilon[X\mapsto T]} = [\![\bigwedge_{i\in I} A_i]\!]_{\varepsilon[X\mapsto T]}$.

Case $\mu X'.A$: In case $X = X'$ and by induction hypothesis $[\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq [\![A]\!]_{\varepsilon[X\mapsto T]}$, $[\![\mu X'.A]\!]_{\varepsilon[X\mapsto S]} = \bigcap\{S' \in \mathscr{P}_{\text{fs}}(\text{STATES}) \mid [\![A]\!]_{\varepsilon[X\mapsto S][X'\mapsto S']} \subseteq S'\} \subseteq [\![\mu X'.A]\!]_{\varepsilon[X\mapsto T]}$. In case $X' \neq X$, then $[\![\mu X'.A]\!]_{\varepsilon[X\mapsto S]} = [\![\mu X'.A]\!]_{\varepsilon}$ and similarly for the set T, thus by reflexivity of $\subseteq$ we conclude the proof case.

□

The least fixpoint operator can be directly expressed in our logic of Section 3. The idea here is simple: we translate a fixpoint into an infinite disjunction that at each step i unrolls the recursion i times. This then semantically corresponds to a limit of an ω-chain generated by a monotonic function, i.e., the least fixpoint.

$$\begin{array}{rcl} \mathsf{expand}_0(\mu X.A) & = & \bot \\ \mathsf{expand}_{i+1}(\mu X.A) & = & \overline{A[\mathsf{expand}_i(\mu X.A)/X]} \\ \overline{\mu X.A} & = & \bigvee_{i\in\omega} \mathsf{expand}_i(\mu X.A) \end{array}$$

$$\overline{X} = X \qquad \overline{\bigwedge_{i\in I} A_i} = \bigwedge_{i\in I} \overline{A_i} \qquad \overline{\neg A} = \neg\overline{A} \qquad \overline{\varphi} = \varphi \qquad \overline{\langle\alpha\rangle A} = \langle\alpha\rangle\overline{A}$$

Note that expand_i is equivariant for all i. Thus, the disjunction in the fixpoint case is well-formed and has support bounded by $\text{supp}(A)$.

Theorem 25. *For any formula A and valuation function ε, $[\![\overline{A}]\!]_{\varepsilon} = [\![A]\!]_{\varepsilon}$.*

Proof. By structural induction over A.

Case $\mu X.A$ We need to show that $[\![\overline{\mu X.A}]\!]_{\varepsilon} = [\![\mu X.A]\!]_{\varepsilon}$. First, we compute the left-hand side to $[\![\overline{\mu X.A}]\!]_{\varepsilon} = [\![\bigvee_{i\in\omega} \mathsf{expand}_i(\mu X.A)]\!]_{\varepsilon} = \bigcup_{i\in\omega}[\![\mathsf{expand}_i(\mu X.A)]\!]_{\varepsilon}$ Second, define $F(S) = [\![A]\!]_{\varepsilon[X\mapsto S]}$, and using this we approximate the fixpoint from below with $\bigcup_{i\in\omega} F^i(\emptyset) = \bigcap\{S \in \mathscr{P}_{\text{fs}}(\text{STATES}) \mid [\![A]\!]_{\varepsilon[X\mapsto S]} \subseteq S\} = [\![\mu X.A]\!]_{\varepsilon}$.

We prove that the above expressions are equal by showing that the elements are equivalent at every step, that is, $[\![\mathsf{expand}_i(\mu X.A)]\!]_{\varepsilon} = F^i(\emptyset)$ for every $i \in \omega$. The proof proceeds by induction on i.

Base case: $[\![\bot]\!]_\varepsilon = \emptyset = F^0(\emptyset)$ by definition. Induction step:

$$\begin{array}{rcl}
[\![\mathsf{expand}_{i+1}(\mu X.A)]\!]_\varepsilon & = & [\![\mathsf{expand}_{i+1}(\mu X.A)]\!]_\varepsilon = \\
& = & [\![\overline{A}[\mathsf{expand}_i(\mu X.A)/X]]\!]_\varepsilon = \\
& = & [\![\overline{A}[\mathsf{expand}_i(\mu X.A)/X]]\!]_\varepsilon = \\
& = & [\![\overline{A}]\!]_{\varepsilon[X\mapsto[\![\mathsf{expand}_i(\mu X.A)]\!]]} \\
& = & [\![\overline{A}]\!]_{\varepsilon[X\mapsto F^i(\emptyset)]} \\
& & \qquad \text{(By induction hypothesis)} \\
& = & [\![A]\!]_{\varepsilon[X\mapsto F^i(\emptyset)]} \\
& & \qquad \text{(By induction hypothesis } [\![A]\!]_\varepsilon = [\![\overline{A}]\!]_\varepsilon \text{ for any } \varepsilon) \\
& = & F(F^i(\emptyset)) \\
& = & F^{i+1}(\emptyset)
\end{array}$$

Other cases are trivial.

□

Proofs from Section 5

Theorem 15

$$P \overset{F/L}{\sim} Q \quad \Longleftrightarrow \quad P \overset{F/L}{=} Q$$

Proof: Direction $\Rightarrow$ is a generalization of Theorem 7.

1. Base case: $A = \langle f\rangle\varphi$ and $f \in F$. Then $f(P) \vdash \varphi$. By static implication $f(Q) \vdash \varphi$, which means $Q \models A$.
2. Inductive step $\langle f\rangle\langle\alpha\rangle A$ where $A \in \mathscr{A}^{F/L}$: Assume $P \models \langle f\rangle\langle\alpha\rangle A$. Then $\exists P' . f(P) \xrightarrow{\alpha} P'$ and $P' \models A$. Without loss of generality we assume also $\mathrm{bn}(\alpha)\#f(Q)$, otherwise just find an alpha-variant of the transition where this holds. Then by simulation $\exists Q' . f(Q) \xrightarrow{\alpha} Q'$ and $P' \overset{L(\alpha,F)/L}{\sim} Q'$. By induction and $P' \models A$ and $A \in \mathscr{A}^{F/L}$ we get $Q' \models A$, whence by definition $Q \models \langle f\rangle\langle\alpha\rangle A$.

The direction $\Leftarrow$ is a generalization of Theorem 10: we prove that $\overset{F/L}{=}$ is an F/L-bisimulation. The modified clauses are:

1. Static implication. Assume $f \in F$, then $f(P) \vdash \varphi$ iff $P \models \langle f\rangle\varphi$ iff $Q \models \langle f\rangle\varphi$ iff $f(Q) \vdash \varphi$.
2. Simulation. The proof is by contradiction. Assume that $\overset{F/L}{=}$ does not satisfy the simulation requirement. Then there exist $f \in F, P, Q, P', \alpha$ such that $P \overset{F/L}{=} Q$ and $f(P) \xrightarrow{\alpha} P'$ and, letting $\mathscr{Q} = \{Q' \mid f(Q) \xrightarrow{\alpha} Q'\}$, for all $Q' \in \mathscr{Q}$ it holds not $P' \overset{L(\alpha,F)/L}{=} Q'$. Choose $\mathrm{bn}(\alpha)\#f(P)$. Thus, for all $Q' \in \mathscr{Q}$ there exists a distinguishing formula in $\mathscr{A}^{L(\alpha,F)/L}$ for P' and Q'. The formula may depend on Q', and by Lemma 16 we can find such a distinguishing formula $B_{Q'} \in \mathscr{A}^{L(\alpha,F)/L}$ for P' and Q' with support in $\mathrm{supp}(P', L(\alpha,F)) \subseteq \mathrm{supp}(P,\alpha,F)$. Therefore the formula

$$B = \bigwedge_{Q'\in\mathscr{Q}} B_{Q'}$$

is well formed in $\mathscr{A}^{L(\alpha,F)/L}$ with support included in $\mathrm{supp}(P',\alpha,F)$. We thus get that $P \models \langle f\rangle\langle\alpha\rangle B$ but not $Q \models \langle f\rangle\langle\alpha\rangle B$, contradicting $P \overset{F/L}{=} Q$. □

Lemma 16 If $A \in \mathscr{A}^{F/L}$ is a distinguishing formula for P and Q, then there exists a distinguishing formula $B \in \mathscr{A}^{F/L}$ for P and Q in such that $\mathrm{supp}(B) \subseteq \mathrm{supp}(P,F)$.

Proof: by direct construction: in the terminology of Pitts [24] ch. 5, B is the conjunction of $\mathrm{hull}_{\mathrm{supp}(P,F)}A$. To spell out the proof: Assume $A \in \mathscr{A}^{F/L}$ and $P \models A$ and not $Q \models A$. Let S be the group of name permutations of names outside $\mathrm{supp}(P,F)$ and let $\mathscr{B}$ be the S-orbit of A, i.e.

$$\mathscr{B} = \{\pi \cdot A \mid \pi \in S\}$$

Clearly $\pi' \cdot \mathscr{B} = \mathscr{B}$ for all $\pi' \in S$, thus $\mathrm{supp}(\mathscr{B}) \subseteq \mathrm{supp}(P,F)$, which means that the formula $B = \bigwedge \mathscr{B}$ is well formed and $\mathrm{supp}(B) \subseteq \mathrm{supp}(P,F)$. By equivariance, if $P \models A$ and $\pi \in S$ then also $P \models \pi \cdot A$. Therefore $P \models B$. Furthermore, since the identity permutation is in S and not $Q \models A$ we get not $Q \models B$. Finally, since L is equivariant we have $\mathrm{supp}(\mathscr{A}^{F/L}) \subseteq \mathrm{supp}(F)$, which means that $\pi \cdot A \in \mathscr{A}^{F/L}$ for all $\pi \in S$, this establishes $B \in \mathscr{A}^{F/L}$. □

Theorem 18 $P \overset{F/L}{\sim} Q$ in $\mathbf{T}$ if and only if $\mathrm{EF}(F,P) \dot{\sim} \mathrm{EF}(F,Q)$ in the L-transform of $\mathbf{T}$.

Proof: For the direction $\Rightarrow$, assume that R is an L-bisimulation. Define R' on the L-transform by including $(\mathrm{EF}(F,P), \mathrm{EF}(F,Q))$ and $(\mathrm{AC}(F,f(P)), \mathrm{AC}(F,f(Q)))$ for all P,Q,f,F such that $f \in F$ and $R_F(P,Q)$. We now prove R' to be a simulation. Assume $R'(S,T)$.

1. Static implication: Assume $S \vdash \varphi$. Then $S = \mathrm{AC}(F,f(P))$ for some F, $f \in F$ and P and $f(P) \vdash \varphi$ holds in $\mathbf{T}$, and $T = \mathrm{AC}(F,f(Q))$ with $R_F(P,Q)$. Thus $f(Q) \vdash \varphi$ whence $T \vdash \varphi$.
2. Simulation: Assume $S \xrightarrow{\alpha} S'$. There are two cases:
   - $S = \mathrm{EF}(F,P) \xrightarrow{f} \mathrm{AC}(F,f(P)) = S'$ and $f \in F$. Then $T = \mathrm{EF}(F,Q)$ where $R_F(P,Q)$. We get $T \xrightarrow{f} \mathrm{AC}(F,f(Q)) = T'$ and $R'(S',T')$. Note here and below that $\mathrm{bn}(f) = \emptyset$.
   - $S = \mathrm{AC}(F,f(P)) \xrightarrow{\alpha} \mathrm{EF}(L(\alpha,F),P') = S'$ and $f(P) \xrightarrow{\alpha} P'$, with $\mathrm{bn}(\alpha)\#\mathrm{AC}(F,f(Q))$. Then also $\mathrm{bn}(\alpha)\#f(Q)$. We get $T = \mathrm{AC}(F,f(Q))$ where $R_F(P,Q)$, so also $f(Q) \xrightarrow{\alpha} Q'$ with $R_{L(\alpha,F)}(P',Q')$. Thus $T \xrightarrow{\alpha} \mathrm{EF}(L(\alpha,F),Q') = T'$, and $R'(S',T')$ as required.

For the direction $\Leftarrow$, assume that R' is a bisimulation in the L-transform of $\mathbf{T}$. Define R_F by $R_F(P,Q)$ if $R'(\mathrm{EF}(F,P), \mathrm{EF}(F,Q))$. We prove R an L-bisimulation. Assume $R_F(P,Q)$.

1. Static implication: Let $f \in F$ and assume $f(P) \vdash \varphi$. Then $\mathrm{EF}(F,P) \xrightarrow{f} \mathrm{AC}(F,f(P))$. Since R' is a bisimulation we get $\mathrm{EF}(F,Q) \xrightarrow{f} \mathrm{AC}(F,f(Q))$. Now $f(P) \vdash \varphi$ means $\mathrm{AC}(F,f(P)) \vdash \varphi$, and again since R' is a bisimulation $\mathrm{AC}(F,f(Q)) \vdash \varphi$, which means $f(Q) \vdash \varphi$ as required.
2. Simulation: Assume $f \in F$ and $f(P) \xrightarrow{\alpha} P'$ with $\mathrm{bn}(\alpha)\#f(Q)$. Without loss of generality additionally assume the transition is represented by an alpha-variant such that $\mathrm{bn}(\alpha)\#F$. We get the transitions
   $$\mathrm{EF}(F,P) \xrightarrow{f} \mathrm{AC}(F,f(P)) \xrightarrow{\alpha} \mathrm{EF}(L(\alpha,F),P')$$
   Since R' is a bisimulation and $\mathrm{bn}(\alpha)\#F, f(Q)$ we get a simulating sequence
   $$\mathrm{EF}(F,Q) \xrightarrow{f} \mathrm{AC}(F,f(Q)) \xrightarrow{\alpha} \mathrm{EF}(L(\alpha,F),Q')$$
   This means that $f(Q) \xrightarrow{\alpha} Q'$ with $R_{L(\alpha,F)}(P',Q')$ as required. □

Recovering a Labelled Semantics for Soft CCP with Local Variables*

Fabio Gadducci
Dipartimento di Informatica,
Università di Pisa, Italy
gadducci@di.unipi.it

Francesco Santini
IIT-CNR,
Pisa, Italy
francesco.santini@iit.cnr.it

Extended Abstract. While the dynamics of a computational system is nowadays more often specified operationally by means of a reduction system (RS), a labelled semantics is needed in order to take advantage of the tools for checking observational properties and equivalences. The methodological advance of the seminal paper [7] is the recognition that labelled semantics should be derivable from reduction semantics in a uniform and systematic way. The proposal of [7] is to build a labelled transition system (LTS) from a RS by identifying labels as the *minimal contexts* needed to trigger a reduction.

Unfortunately, this notion of minimality, called *relative pushout*, proved to be difficult to check and apply, also for most basic calculi such as CCS [4, 8]. A recent proposal [5] suggests to remove the requirement of minimality, in order to characterize a class of LTSs verifying weaker coherence properties, yet ensuring the adequacy of the observational equivalence.

One of the testbed of the proposal has been *Concurrent Constraint Programming* (CCP) [9], a language based on a shared-memory communication pattern: processes interact by either posting or checking partial information, represented as constraints in a global store. CCP belongs to the larger family of process calculi, thus a syntax-driven operational semantics represents the computational steps. For example, the term **tell**(c) is the process that posts c in the store, and the term **ask**(c) $\rightarrow$ P is the process that executes P if c can be derived from the information in the store.

The formalism is parametric with respect to the entailment relation. Under the name of *constraint system*, the information recorded on the store is structured as a partial order (actually, a lattice) $\leq$, where $c \leq d$ means that c can be derived from d. Under a few requirements over such systems, CCP has been provided with (coincident) operational and denotational semantics.

A key aspect of CCP is the *idempotency* of constraint composition: adding an information twice does not change the store. In the soft variant of the formalism (Soft CCP, SCCP [2]), constraint systems may distinguish the number of occurrences of a piece of information. Dropping idempotency requires a full reworking of the theory. Although an operational semantics for SCCP has been devised [2], neither the denotational nor the labelled one has been reintroduced.

The work in [9] establishes a denotational semantics for CCP and an equational theory for infinite agents. More recently, in [1] the authors prove that the axiomatisation is underlying a specific weak bisimilarity among agents, thus providing a clear operational understanding. The key ingredients are a complete lattice as the domain of the store, with least upper bound for constraint combination, and a notion of compactness such that domain equations for the parallel composition of recursive agents are well-defined. The soft version [2] drops the upper bound for combination in exchange of a monoidal operator. Thus, the domain is just a (not

*Research partially supported by the MIUR PRIN 2010LHT4KM CINA and 2010XSEMLC "Security Horizons".

Submitted to:
MeMo 2015

$$\frac{\langle A,\sigma' \otimes \exists_x \sigma_0\rangle \longrightarrow \langle B,\sigma''\rangle \quad \textit{with } \sigma_0 = \sigma \ominus \exists_x \sigma'}{\langle \exists_x^{\sigma'} A,\sigma\rangle \longrightarrow \langle \exists_x^{\sigma_1} B,\sigma_0 \otimes \exists_x \sigma_1\rangle \quad \textit{with } \sigma_1 = \sigma'' \ominus \exists_x \sigma_0} \textbf{ Hide}$$

$$\frac{\langle A,\sigma' \otimes \sigma_0[^z/_x]\rangle \xrightarrow{\alpha} \langle B,\sigma''\rangle \quad \textit{with } \sigma_0 = \sigma \ominus \exists_x \sigma', x \notin sv(\alpha), z \notin fv(A) \cup sv(\sigma) \cup sv(\sigma')}{\langle \exists_x^{\sigma'} A,\sigma\rangle \xrightarrow{\alpha[^x/_z]} \langle \exists_x^{\sigma_1} B,\sigma_0 \otimes \alpha[^x/_z] \otimes \exists_x \sigma_1\rangle \quad \textit{with } \sigma_1 = \sigma'' \ominus (\alpha \otimes \sigma_0[^z/_x])} \textbf{ L-Hide}$$

Table 1: Unlabelled and labelled version of the rule for the local hiding of variables.

necessarily complete) partial order, possibly with finite meets and a residuation operator (a kind of inverse of the monoidal one) in order to account for algorithms concerning constraint propagation. Indeed, the main use of SCCP has been in the generalisation of classical constraint satisfaction problems, hence the lack of investigation about denotational semantics.

In [6] we connected the works on the soft [2] and the classical (also indicated in the literature as "crisp") [1, 9] paradigm by investigating a labelled (and an unlabelled) semantics for a deterministic fragment of SCCP. In particular, the result was a mix of those investigated in the two communities, namely, a monoid whose underlying set of elements form a complete lattice. Residuation theory provided an elegant way to define a weak inverse operator of tensor $\otimes$ with the purpose to determine the minimal information that enables the firing of actions in the LTS, thus distilling a suitable notion of labelled transition.

The operational semantics chosen in [6] favoured a global view of variables, namely, the agent $\exists_x A$, roughly corresponding to an existential quantification over the variable x of the agent A, would evolve to the agent $A[^y/_x]$, for y a fresh variable (with respect to the agent A and the store it is valued in). In this work we consider instead local variables, where the hiding operator carries some information on the variables it abstracts. More precisely, according to [3] we consider an extended operator $\exists_x^\sigma$, for σ the local store. Thanks again to the residuation operator, the rule for the extended hidings can be defined as **Hide** in Tab. 1. The intuition is that variable x may be local to a component $\exists_x \sigma'$ of the store σ, yet visible at a global level: we must then evaluate A in the store when the local x is hidden, yet the possible duplications are removed (e.g., $\exists_x \sigma'$ may already occur in the global store σ). The final store intuitively contains in σ_1 the original σ' increased by what has been added by the step.

In the labelled version of the rule (**L-Hide** in Tab. 1) we rename the global x with a fresh variable z, instead of hiding x in the global store, as we do in the corresponding unlabelled rule. We accomplish this in order to keep track of the global x in α: finally, in the result we restore the occurrences of z back to x.

The work is rounded up by a preliminary correspondence results between fair computations and bisimulation for soft CCP with local variables.

References

[1] Andrés Aristizábal, Filippo Bonchi, Catuscia Palamidessi, Luis Fernando Pino & Frank D. Valencia (2011): *Deriving labels and bisimilarity for concurrent constraint programming*. In Martin Hofmann, editor: *FOSSACS 2011*, *LNCS* 6604, Springer, pp. 138–152.

[2] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (2006): *Soft concurrent constraint programming*. *ACM ToCL* 7(3), pp. 563–589.

[3] Frank S. de Boer, Maurizio Gabbrielli, Elena Marchiori & Catuscia Palamidessi (1997): *Proving concurrent constraint programs correct*. *ACM ToPLaS* 19(5), pp. 685–725.

[4] Filippo Bonchi, Fabio Gadducci & Barbara König (2009): *Synthesising CCS bisimulation using graph rewriting*. *I& C* 207(1), pp. 14–40.

[5] Filippo Bonchi, Fabio Gadducci & Giacoma Valentina Monreale (2014): *A general theory of barbs, contexts, and labels*. *ACM ToCL* 15(4), pp. 35:1–35:27.

[6] Fabio Gadducci, Luis Pino, Francesco Santini & Frank Valencia (2015): *A labelled semantics for soft CCP*. In T. Holvoet & M. Viroli, editors: *COORDINATION*, *LNCS* 9037, pp. 133–149.

[7] James J. Leifer & Robin Milner (2000): *Deriving bisimulation congruences for reactive systems*. In Catuscia Palamidessi, editor: *CONCUR 2000*, LNCS, pp. 243–258.

[8] Robin Milner (2006): *Pure bigraphs: Structure and dynamics*. *I&C* 204(1), pp. 60–122.

[9] Vijay A. Saraswat, Martin C. Rinard & Prakash Panangaden (1991): *Semantic foundations of concurrent constraint programming*. In David S. Wise, editor: *POPL 1991*, ACM Press, pp. 333–352.