Axiomatizations from Structural Operational Semantics: Theory and Tools

Eugen Ioan Goriac
Doctor of Philosophy
August 2013
School of Computer Science
Reykjavík University

Ph.D. DISSERTATION

ISSN 1670-8539

Axiomatizations from Structural Operational Semantics: Theory and Tools

by

Eugen Ioan Goriac

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

August 2013

Thesis Committee:

Luca Aceto, Supervisor
Prof., Reykjavík University

Jos Baeten, Examiner
Prof., Eindhoven University of Technology

Anna Ingólfsdóttir
Prof., Reykjavík University

MohammadReza Mousavi
Prof., Halmstad University

Axiomatizations from Structural Operational Semantics: Theory and Tools

Eugen Ioan Goriac

August 2013

Abstract

Structural Operational Semantics (SOS) is a well known standard for specifying language semantics in a natural, yet rigorous way. Once a formal way of checking for the equivalence of two programs written in such a language is provided, it is of great interest to derive efficient automated methods to prove if equivalences hold. Also of high interest for language designers is the possibility of enhancing the expressiveness of SOS in a formal manner, preserving as much from the already developed meta-theory of SOS as possible. The thesis focuses on these two areas, both from a theoretical and a practical perspective.

The line of research addresses the extension of SOS with predicates and data, while lifting certain results from the meta-theory of SOS to these extensions. These results include automatically deriving axiomatizations for reasoning on program equivalence, and checking for compliance to rule formats in order to guarantee desired properties. Besides these extensions, the thesis provides an axiomatization for the coordination language Linda, presents a method to optimize axiomatizations for language constructs that are commutative, and presents a rule format for idempotent unary operators and idempotent terms.

The practical aspect of this thesis consists of a core software framework for working with SOS meta-theories, named Meta SOS, which is implemented in Maude. The framework includes components for automatically deriving axiomatizations, performing simulations, and checking whether language constructs comply to a format for commutativity. It is designed in a modular and extensible fashion, and serves as a base for future implementations of other results from the meta-theory of SOS.

Axiomatizations from Structural Operational Semantics: Theory and Tools

Eugen Ioan Goriac

Ágúst 2013

Útdráttur

Þegar þróa á áreiðanlegan og stöðugan hugbúnaðar er oft fyrsta skerfið að lýsa á formlegan hátt hvað skilyrðum hann á að uppfylla. Þetta er gert með því að búa til formleg líkön af hugbúnaðinum sem nota má í þesum tilgangi. Það hefur verið vinsælt á síðari árum að nota líkön sem byggja a svokallaða uppbyggingarvinnslumerkingafræði, (á ensku "structural operational semantics en oftast vitnað til sem SOS) sem er fomlegur fræðilegur rammi (e. meta theory) til að lýsa eiginleikum formlega málsins sem kerfinu er lýst í.

Á síðastliðnum árum hafa miklar rannsóknuir verið stundaðar og gera niðurstöður þeirra það mögulegt að segja til um eiginleika formlegra mála með því að líta á reglurnar sem lýsa merkingarfræði þeirra.

I þessu doktorverkefni leggur höfundur sitt af mörkum til SOS fræðanna frá tveimur sjónarmiðum. Annars vegar hefur hann útvíkkað almennu fræðin með því að bæta við mikilvægum hugtökum sem ekki hefur verið fjallað um áður. Hins vegar hefur hann forritað kerfi, byggt á almennu fræðunum, þar sem hægt er að spyrja spurninga um ákveðin tilvik og fá svör við þeim. Þetta kerfi hefur fengið góðar undirtektir hjá væntanlegum notendum og er mikilvægt skref í áttina að nýtingu á rannsóknaniðurstöðum í greininni.

Acknowledgements

The first two persons I would like to thank are my supervisors and collaborators, Anna Ingólfsdóttir and Luca Aceto. I am more than grateful for the freedom and the support they gave me during the period of earning my PhD. They helped creating the ideal environment both for my professional and personal evolution.

I thank my collaborators without whom this work would have never been possible Daniel Gebler, Georgiana Caltais, Michel Reniers, MohammadReza Mousavi. Besides my supervisors and MohammadReza Mousavi, the Thesis Committee had Jos Baeten as the examiner. To all of them I am truly thankful for their thoughtful comments and suggestions.

I am thankful to the friends I made abroad who helped me broaden my perspective of life Alexander, Alexandra, Ali, Anaïs, Andrea, Angelo, Annemie, Aron, Bjarki, Caro, Cindy, Claudia, Claudio, Dario, Davið, Elizes, Filippo, Gabe, Gabriel, Hamid, Ileana, Ioana, Iuliana, Jacky, Jan, Joe, Jolanda, Juan, Karolina, Kári, Lilja, Lyuba, Maarja, Marcello, Maria, Marijke, Marketa, Matteo, Mădălina, Natalia, Neil, Niccolò, Nicola, Óli, Patrycja, Pradipta, Raluca, Sirrý, Skarpi, Stephan, Tanja, Ute, Verena, Verity, Valla, Victor, Viktor, Viky. I address special thanks to Andrei Manolescu, Deepa Iyengar, Ingibergur Þorkelsson and Marjan Sirjani, who provided me with support in key moments. I thank those back home for helping me preserve a wonderful sense of belonging Adriana, Amalia, Andrei, Constantin, Doru, Georgiana, Iulian, Leti, Măriuca, Micsandra, Mihaela, Mihai, Mimi, Radu, Ştefan.

I am deeply grateful to my tutors Liliana Ruset, Henri Luchian, Ovidiu Gheorghieş and Dorel Lucanu, who tremendously influenced my evolution in the field of computer science. Last, but not least, I thank my parents Carmen and Viorel for their love, trust and unconditional support.

The work in this thesis was partially supported by the projects 'Meta-theory of Algebraic Process Theories' (nr. 100014021) and 'Extending and Axiomatizing Structural Operational Semantics: Theory and Tools' (nr. 1102940061) of the Icelandic Research Fund.

Contents

Chapter 1

Introduction

Specifying and analyzing the behaviour of operating systems, communication protocols, and embedded systems, among others, have always been both of great interest and challenging in computer science. The aforementioned computer systems are but a few examples of *concurrent reactive systems* [4], which are generally thought of as hardware and/or software devices that compute by reacting to stimuli from their environment.

The interactive nature of reactive systems makes them particularly difficult to develop. It is therefore not surprising that a substantial research effort has been devoted to the development of formal aproaches for modelling, specification and verification of such systems.

1.1 Process Algebras

Over the last thirty years, *process algebras* [25, 33, 95, 106] have been actively and efficiently used for the formal specification and verification of concurrent reactive systems [31]. Their approach involves associating a mathematical object, referred to as *process*, to each reactive system. The formal analysis of processes usually refers to checking whether an implementation complies to its specification, or to verifying logical properties processes satisfy. The main two approaches for formal analysis are *equivalence checking*, which we will focus on in this thesis, and *model checking*. The former relies on logic reasoning, while the latter involves checking for properties by means of exhaustive searches within the state space of a process evolution.

A process algebra is often given by defining a language *semantics* for describing processes, a notion of *behavioural equivalence* over processes, and a set of *axioms* for deriving process equivalences using equational logic. One could also consider a *preorder* relation between processes instead of a behavioural equivalence. Preorders are often used to describe formally when the behaviour of one process is an "approximation of" that of another one. In this thesis, we shall focus on behavioral equivalences and we do not deal with preorders.

It is crucial that the chosen notion of behavioural equivalence is a *congruence* because this means that, given a set of equivalent processes, it holds that a context with a placeholder for one of these processes will present the same behaviour, independently of the "plugged" process. This has high practical value as it implies that, for instance, local optimizations do not alter the global behaviour, and lead to global optimizations.

Once a suitable notion of behavioural congruence has been identified, it is natural to ask oneself what are the "laws of programming" that hold with respect to it. Such laws can be expressed in a clear and concise way by means of equations, or axioms. For instance, the expectation that a parallel composition operator $\|$ be associative is captured by the axiom

$$(x \,\|\, y) \,\|\, z = x \,\|\, (y \,\|\, z).$$

It is mandatory that the axioms be *sound*, so that all the equalities between processes that can be proved from the axioms are valid behavioural equivalences. Ideally, the axioms should also be *complete*, in order to guarantee that any valid equivalence between two processes can be formally inferred from the equations. There also exist *ground-complete* axiomatizations, which can only be used for fully specified processes, in contrast to pure complete axiomatizations that are given for generic processes with yet unspecified components.

The true power of sound and (ground-)complete axiomatizations consists in their ability to allow for syntactic reasonings on the behaviour of a process without generating its whole state space. This may help in combating the state explosion problem and in the analysis of infinite-state systems. Moreover, (ground-)complete axiomatizations of process equivalences capture their essence by means of a collection of laws. This often allows one to highlight the differences between two different equivalences in terms of a few revealing axioms.

Traditionally, language semantics has been given in an axiomatic, denotational or operational manner [33]. Three representative process algebras, one for each class of semantics, are ACP [25], CSP [95] and CCS [106].

In his early work on CCS, in order to provide operational semantics, Milner introduced the idea of associating a labelled transition system to a process term. One systematic way to obtain labelled transition systems from terms in a process description language is via Structural Operational Semantics (SOS) [124].

1.2 Structural Operational Semantics

SOS was introduced more than thirty years ago by Matthew Hennessy and Gordon Plotkin as a systematic way to assign operational semantics to programming languages by means of a set of inference rules [93]. As stated by Plotkin in his account of the history of the ideas leading to SOS [123],

> . . . structural operational semantics was intended as being like an abstract machine but without all the complex machinery in the configurations, just the minimum needed to explain the semantic aspects of the programming language constructs. The extra machinery is avoided by the use of the rules, making the exploration of syntactic structure implicit rather than drearily explicit.

Operational semantics characterizes the execution of a process by defining the *transitions* it can perform. Each transition of a process may carry a *label*, which describes abstractly the computational step that led to it. Such a label may, for instance, stand for the communication of a message to the "outside world". A *positive transition formula* is a triple written as $P \xrightarrow{l} P'$, where P and P' denote the process states before and, respectively, after the transition, while l is the label of the transition. There also exist *negative transition formulas*, which are pairs consisting of a process state P and a label l and are written $P \overset{l}{\nrightarrow}$. Such a formula has the interpretation that process P cannot perform the transition with label l.

In SOS, a set of transitions is defined implicitly by means of a collection of syntax-driven *rules*, which have the form $\frac{premises}{conclusion}$. The *premises* are a (possibly empty) set of transition formulas. If they are satisfied, then the *conclusion*, which is a positive transition formula, holds.

A formalizations of SOS is given by the so-called *Transition System Specifications* (TSS's), which were first introduced in [86]. Intuitively, a TSS consists of a signature, which describes the constructs in the language to which one is giving semantics, together with their arities, and of a collection of inference rules. The collection of inference rules is used to specify the set of legal transitions in the semantics of a language. Intuitively, a transition is legal if, and only if, its existence can be justified using the rules.

In order to concretely see how a TSS is provided, consider the following example of, BCCSP [81], a basic concurrent language for the description of nondeterministic processes.

Example 1.2.1. *The syntax of the language consists of a constant* $\mathbf{0}$ (deadlock)*, the binary operator* $\_+\_$ (nondeterministic choice)*, and the unary operators* $\alpha.\_$ (action prefix)*, where* α *ranges over a finite set of action labels* $\mathcal{A}$*. Formally, the grammar of the language is:*

$$P ::= \mathbf{0} \mid \alpha.P \mid P + P,$$

where α *is from* $\mathcal{A}$*.*

Intuitively, $\mathbf{0}$ *represents a process that does not exhibit any behaviour,* $P_1 + P_2$ *behaves either like* P_1 *or* P_2*, and* $\alpha.P$ *is a process that first performs action* α *and then behaves like* P*. Formally, the behaviour of these operators is given by the following set of SOS rules, for every* α *in* $\mathcal{A}$*:*

$$(1_\alpha)\ \frac{}{\alpha.x \xrightarrow{\alpha} x} \qquad (2_\alpha)\ \frac{x \xrightarrow{\alpha} x'}{x + y \xrightarrow{\alpha} x'} \qquad (3_\alpha)\ \frac{y \xrightarrow{\alpha} y'}{x + y \xrightarrow{\alpha} y'}.$$

Let us analyze how these rules are used in order to derive the evolution of a concrete process. Assume that $\mathcal{A} = \{a, b\}$*. This means that every rule has two instantiations – one when* α *is* a*, and the other when* α *is* b*. Consider the process term* $a.(\mathbf{0} + b.\mathbf{0})$*. It only matches* $a.x$*, the left hand side of the conclusion of rule* (1_a)*, by mapping* x *to* $\mathbf{0} + b.\mathbf{0}$*. There are no premises to be satisfied for this rule, therefore the following transition takes place:* $a.(\mathbf{0} + b.\mathbf{0}) \xrightarrow{a} \mathbf{0} + b.\mathbf{0}$*. The resulting term does match the conclusion of rules* (2_a) *and* (2_b)*, but neither of these rules has a satisfiable premise, as* x *is mapped to* $\mathbf{0}$ *and there exists no rule that can derive a transition for* $\mathbf{0}$*. The only rule that can be fired for* $\mathbf{0} + b.\mathbf{0}$ *is* (3_b)*. By mapping* y *to* $b.\mathbf{0}$*, this rule's premise is satisfied because, according to rule* (1_b)*, it holds that* $b.\mathbf{0} \xrightarrow{b} \mathbf{0}$*. Therefore* y' *is mapped to* $\mathbf{0}$*, which leads, overall, to the following transition:* $\mathbf{0} + b.\mathbf{0} \xrightarrow{b} \mathbf{0}$*.*

Given a TSS, it is of great interest to be able to verify whether two different processes behave in the same way. One may want to check, for instance, if a process behaves according to a given specification, which, in turn, is another (usually more "abstract") process. It may also be the case that a process is meant to optimize another one, and that the implementer wants to make sure that both processes perform similarly. We need, therefore, a way of defining what "similar behaviour" between processes means. Let us consider the following notion of process behaviour equivalence, defined in an informal manner.

Informal Definition 1.2.2 (Bisimilarity)**.** *Two processes S and T are* bisimilar *if, and only if,*

- *if S can perform an α labelled transition to a process S′, then it must hold that T performs an α-labelled transition to a process T′, such that S′ and T′ are bisimilar, and*
- *if T can perform an α labelled transition to a process T′, then it must hold that S performs an α-labelled transition to a process S′, such that T′ and S′ are bisimilar.*

The formal definitions of bisimilarity and some of its variants are given throughout the following chapters. The informal definition above, however, suffices for the purpose of this introduction.

It is important to note that, according to Definition 1.2.2, $\mathbf{0}$ is bisimilar with $\mathbf{0}$. This is because $\mathbf{0}$ cannot perform any action, which means that both conditions hold by default.

In order to get a sense of how important it is to work with a formal notion of equivalence, consider that the length of a process is the number of characters required to specify it. Also, let us assume that the shorter a process is, the higher interest it presents. Therefore, when given a process, we want to know whether there is a shorter one that exhibits the same behaviour. Considering again the process $a.(\mathbf{0} + b.\mathbf{0})$, we now become interested to check if there exists some other shorter, but equivalent process.

We have previously shown that $a.(\mathbf{0} + b.\mathbf{0}) \xrightarrow{a} \mathbf{0} + b.\mathbf{0} \xrightarrow{b} \mathbf{0}$. It is easy to check that the term $a.b.\mathbf{0}$ evolves in this manner: $a.b.\mathbf{0} \xrightarrow{a} b.\mathbf{0} \xrightarrow{b} \mathbf{0}$. As previously stated, $\mathbf{0}$ is bisimilar with $\mathbf{0}$. Both $\mathbf{0} + b.\mathbf{0}$ and $b.\mathbf{0}$ can only perform a b-labelled transition, and the two resulting processes are bisimilar, which means that $\mathbf{0} + b.\mathbf{0}$ and $b.\mathbf{0}$ are bisimilar. We infer, in a similar fashion that processes $a.(\mathbf{0} + b.\mathbf{0})$ and $a.b.\mathbf{0}$, are bisimilar. Therefore $a.b.\mathbf{0}$ is a shorter process than $a.(\mathbf{0} + b.\mathbf{0})$, with equivalent behaviour.

1.3 Meta-theory of SOS

By imposing syntactic restrictions on the form of the rules in a TSS, several interesting semantic properties, such as well-definedness and finite branching of the transition relation [50, 82, 84], congruence properties for behavioural equivalences and preorders [47, 48, 80, 86], operational conservativity [13, 70], and issues related to security [134] and probability [37, 98], can be inferred by purely syntactic means for the corresponding operational semantics. These restrictions on the form of the inference rules used in TSS's give rise to the so-called *rule formats*, which constitute the basis on which a *meta-theory* of SOS has been developed over the last twenty years. (See the survey papers [14] and [118] for information on results in the meta-theory of SOS.) The development of the meta-theory of SOS allows for the generalization of well known results in the field of process algebras to classes of languages, namely all those whose operational specification is given in terms of rules fitting a rule format. A rule format is a syntactic template for rules of the form $\frac{premises}{conclusion}$ with the intent that, if the rules for an operation/language match the specified format, then the operation/language has some given semantic properties.

We have already seen why it is important for a behavioural equivalence to be a congruence. Proofs of congruence for bisimilarity, for instance, have been given for many process algebras. Naturally, these proofs vary from language to language, but their structure is, essentially, the same. Rule formats have proved to be very useful in integrating the essence of these proofs. In order to give a concrete example of a rule format, let us now present the one introduced in [49], referred to as GSOS. This format is often used throughout the thesis and its importance lies, among other things, in the guarantee that bisimilarity is a congruence for every system that complies to the GSOS restrictions. The format also guarantees that the induced transition systems are computable and finitely branching.

Definition 1.3.1 ([49]). *A deduction rule for an operator f of arity n is in the GSOS format if, and only if, it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \overset{l_{ik}}{\nrightarrow} \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{l} t}$$

where, for every $1 \leq i \leq n$, m_i and n_i are natural numbers, and for every $1 \leq j \leq m_i, 1 \leq k \leq n_i$, x_i's and y_{ij}'s are all distinct variables, l_{ij}'s, l_{ik}'s and l are labels, and t is a process term with variables including at most the x_i's and the y_{ij}'s as variables.

A TSS is in the GSOS format when it has a finite number of operations, a finite set of labels, a finite set of deduction rules, and all its deduction rules are in the GSOS format. We shall sometimes refer to a TSS in the GSOS format as a GSOS system.
Theorem 1.3.2 ([49])**.** *Bisimilarity is a congruence for every GSOS system.*

It is easy to check that BCCSP is a GSOS system, and therefore, by Theorem 1.3.2, bisimilarity is a congruence for BCCSP.

Other formats for congruence are given in [30, 84, 86, 116, 132, 138].

Alongside the aforementioned results in the meta-theory of SOS, there has lately been much progress on the automated generation of sound and ground-complete *axiomatizations* of languages defined by SOS [2, 6, 35]. Instead of checking for the equivalence of two processes by directly using the definition, one can, instead, check for it at a purely syntactic level, using axiomatizations. Consider, for instance, the following axiomatization.

$$\begin{array}{rcll} x + y & = & y + x & \text{(commutativity)} \\ (x + y) + z & = & x + (y + z) & \text{(associativity)} \\ x + x & = & x & \text{(idempotence)} \\ x + \mathbf{0} & = & x & \text{(unit element)} \end{array}$$

It is well known that this axiomatization is sound and ground-complete for bisimilarity over BCCSP [106]. Due to this fact, we can syntactically show that $a.(\mathbf{0} + b.\mathbf{0})$ is bisimilar to $a.b.\mathbf{0}$. The reasoning involves using the commutativity and unit element axioms in order to derive that $\mathbf{0} + b.\mathbf{0} = b.\mathbf{0}$. Recall that bisimilarity is a congruence for BCCSP. We can, therefore, use the fact that $\mathbf{0} + b.\mathbf{0} = b.\mathbf{0}$ inside the context $a.[\_]$. We thus infer that $a.(\mathbf{0} + b.\mathbf{0}) = a.b.\mathbf{0}$, which means, by the soundness of the axiomatization, that $a.(\mathbf{0} + b.\mathbf{0})$ and $a.b.\mathbf{0}$ are bisimilar.

Another important aspect related to the specification of concurrent reactive systems is the *expressiveness* of process definition languages. For some types of systems it may be the case that the executional semantics given by the standard SOS framework, surveyed in [14, 118] is not expressive enough. This happens when we need the process states to be characterized by certain properties. A process state may, for instance, satisfy predicates (such as termination) [19], have associated data and store (memory) [115, 116], or be characterized by nominal aspects [72]. The more constructs are syntactically accepted by operational semantics deduction rules, the higher is the complexity of systems that can be specified with the obtained languages.

1.4 Software tools

Though using the results from the meta-theory of SOS tremendously simplifies the work of language designers, doing it "by hand" can still be tedious and error-prone. Therefore it is of high interest to have software tools that can automatically apply these results, allowing designers to focus more on the language design itself.

To our knowledge, the development of dedicated software solutions for language design and analysis that rely on the meta-theory of SOS is currently in its infancy. Perhaps the most notable effort towards such a solution is the prototype presented in [59]. The prototype facilitates the specification of SOS languages and features performing simulations, as well as checking for processes bisimilarity and for the specification compliance to the GSOS format. LETOS [87] is a lightweight tool that makes some first steps towards checking operational conservativity along the lines proposed in the paper [86]. The Maude MSOS Tool (MMT) [57] also allows the user to define SOS for languages based on the Modular SOS framework of Mosses [108], however it does not focus on rule formats. None of these tools provides support for deriving axiomatizations in order to perform algebraic reasoning. The Process Algebra Manipulator (PAM) [99] does feature algebraic reasoning, but only for a limited number of languages, namely CCS [106], CSP [95] and LOTOS [54].

1.5 Contributions

The thesis focuses on using SOS rule formats in order to automatically derive properties and axiomatizations for certain classes of systems.

The remainder of this section presents the structure of the thesis, concretely enumerating the contributions and indicating the papers they are based on. First come all the chapters with theoretical contributions. Then there are two chapters presenting software tools that implement results from the meta-theory of SOS, some of which are obtained in this thesis.

Theory

- **A Ground-Complete Axiomatization of Stateless Bisimilarity over Linda.** Chapter 2 offers a finite, ground-complete axiomatization of a notion of

bisimilarity with data, named stateless bisimilarity [116], over the tuple-space-based coordination language Linda [56, 78]. As stepping stones towards this result, the chapter provides axiomatizations of stateless bisimilarity over the sequential fragment of Linda without the *nask* primitive, which tests for the absence of a tuple in the tuple space, and over the full sequential sub-language. It is also shown that stateless bisimilarity coincides with standard bisimilarity over the sequential fragment of Linda without the *nask* primitive. The material contained in this chapter is based on the submitted paper [15].

- **Axiomatizing GSOS with Predicates.** Chapter 3 introduces an extension of the GSOS rule format with predicates. This format is a basis for generalizing the technique proposed by Aceto, Bloom and Vaandrager for the automatic generation of ground-complete axiomatizations of bisimilarity over GSOS systems. This paves the way to checking strong bisimilarity over process terms by means of theorem-proving techniques. The material contained in this chapter is based on the content of the published paper [7].

- **Algebraic Meta-Theory of Processes with Data.** There exists a rich literature of rule formats guaranteeing different algebraic properties for formalisms with a SOS. Moreover, there exist a few approaches for automatically deriving axiomatizations characterizing strong bisimilarity of processes. This literature has never been extended to the setting with data (e.g. to model storage and memory). Chapter 4 shows how the rule formats for algebraic properties can be exploited in a generic manner in the setting with data. Moreover, it introduces a new approach for deriving sound and ground-complete axiom schemata for stateless bisimilarity, based on intuitive auxiliary function symbols for handling the store component. We do restrict, however, the axiomatization to the setting where the store component is only given in terms of constants. The material contained in this chapter is based on the content of the published paper [77].

- **Exploiting Algebraic Laws to Improve Mechanized Axiomatizations.** In the field of SOS, there have been several proposals both for syntactic rule formats guaranteeing the validity of algebraic laws, and for algorithms for automatically generating ground-complete axiomatizations. However, there has been no synergy between these two types of results. Chapter 5 takes the first steps in marrying these two areas of research in the meta-theory of SOS and shows that taking algebraic laws into account in the mechan-

ical generation of axiomatizations results in simpler axiomatizations. The proposed theory is applied to a paradigmatic example from the literature, showing that, in this case, the generated axiomatization coincides with a classic hand-crafted one. The material contained in this chapter is based on the content of the published paper [18].

- **SOS Rule Formats for Idempotent Terms and Idempotent Unary Operators.** A unary operator f is idempotent if the equation $f(x) = f(f(x))$ holds. On the other end, an element a of an algebra is said to be an idempotent for a binary operator $\odot$ if $a = a \odot a$. Chapter 6 presents a rule format for SOS that guarantees that a unary operator be idempotent modulo bisimilarity. The proposed rule format relies on a companion one ensuring that certain terms are idempotent with respect to some binary operator. This study also offers a variety of examples showing the applicability of both formats. The material contained in this chapter is based on the content of the published papers [17] and [21].

Software tools

- **PREG Axiomatizer – A Ground Bisimilarity Checker for GSOS with Predicates.** In Chapter 7 we present PREG Axiomatizer, a tool used for proving strong bisimilarity between ground terms consisting of operations in the GSOS format extended with predicates. It automatically derives sound and ground-complete axiomatizations using the technique proposed in Chapter 3. These axiomatizations are provided as input to the Maude system [59], which, in turn, is used as a reduction engine for provided ground terms. These terms are bisimilar if, and only if, their normal forms obtained in this fashion are equal. The motivation of this tool is the optimized handling of equivalence checking between complex ground terms within automated provers and checkers. The material contained in this chapter is based on the content of the published paper [8]. The tool is downloadable from `http://goriac.info/tools/preg-axiomatizer/`.

- **Meta SOS – A Maude Based SOS Meta-Theory Framework.** Chapter 8 is devoted to the presentation of Meta SOS, a software framework designed to integrate the results from the meta-theory of SOS. These results include deriving semantic properties of language constructs just by syntactically analyzing their rule-based definition, as well as automatically deriving sound and ground-complete axiomatizations for languages, when considering a

notion of behavioural equivalence. This chapter describes the Meta SOS framework by blending aspects from the meta-theory of SOS, details on their implementation in Maude [59], and running examples. The material contained in this chapter is based on the content of the published paper [16]. The framework is downloadable from http://goriac.info/tools/meta-sos/.

Chapters 2-8 are self-contained, as they are adaptations of the papers on which they are based. Compared to the paper it is based on, each chapter has at least some minor changes which resulted when adapting it to the context and format of the thesis, or correcting minor errors spotted after publication. The chapters that have more significant improvements clearly state this in the end of their introduction. Though the notation has slightly evolved over time, each chapter preserves the notation used when its related paper was conceived. For this reason different chapters may have different notations for the same concept.

Chapter 9 is a conclusion of the thesis contributions and presents possible future lines of development and research.

1.5.1 Publications resulting while working on the thesis

Below is a list of publications produced while working on the Ph.D. thesis.

1. Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. Axiomatizing GSOS with predicates. In *Proceedings of the Eighth Workshop on Structural Operational Semantics*, volume 62 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–15. 2011.

2. Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. PREG Axiomatizer – a ground bisimilarity checker for GSOS with predicates. In *Proceedings of the 4th Conference on Algebra and Coalgebra in Computer Science (CALCO 2011)*, volume 6859 of Lecture Notes in Computer Science, pages 378–385. 2011.

3. Luca Aceto, Eugen-Ioan Goriac, Anna Ingólfsdóttir, Mohammad Reza Mousavi, and Michel Reniers. Exploiting algebraic laws to improve mechanized axiomatizations. In *Proceedings of the 5th Conference on Algebra and Coalgebra in Computer Science (CALCO 2013)*, Lecture Notes in Computer Science. To appear. 2013.

4. Luca Aceto, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. SOS rule formats for idempotent terms and idempotent unary operators. In *Proceedings of the 39th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2013)*, volume 7741 of *Lecture Notes in Computer Science*, pages 108–120. 2013.

5. Luca Aceto, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. SOS rule formats for idempotent terms and idempotent unary operators. In *Journal of Logic and Algebraic Programming*. To Appear. 2013

6. Luca Aceto, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. A ground-complete axiomatization of stateless bisimilarity over Linda. Technical report, Reykjavik University, 2013. Submitted to *Information Processing Letters*. Available from http://www.ru.is/faculty/luca/PAPERS/axiomatizing_linda.pdf.

7. Daniel Gebler, Eugen-Ioan Goriac, and Mohammad Reza Mousavi. Algebraic meta-theory of processes with data. In *Proceedings of the Tenth Workshop on Structural Operational Semantics*. To appear. 2013.

8. Luca Aceto, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. Meta SOS – a Maude based SOS meta-theory framework. In *Proceedings of the Tenth Workshop on Structural Operational Semantics*. To appear. 2013.

9. Marcello M. Bonsangue, Georgiana Caltais, Eugen-Ioan Goriac, Dorel Lucanu, Jan J. M. M. Rutten, and Alexandra Silva. Automatic equivalence proofs for non-deterministic coalgebras. In *Science of Computer Programming*. To appear. 2013.

Paper 9 does not concern SOS, therefore the results presented there are not part of the present thesis.

Chapter 2

A Ground-Complete Axiomatization of Stateless Bisimilarity over Linda

2.1 Introduction

The goal of this chapter is to contribute to the study of equational axiomatizations of behavioural equivalences for processes with data—see, e.g.,the references [64, 89, 90, 91] for earlier contributions to this line of research. Specifically, we present a ground-complete axiomatization of stateless bisimilarity from [52, 62, 85, 116] over the well-known tuple-space-based coordination language Linda [56, 78].

Linda is a, by now classic, example from a family of coordination languages that focus on the explicit control of interactions between parallel processes. A communication between Linda processes takes place by accessing tuples in a shared memory, called the *tuple space*, which is a multiset of tuples. The communication mechanism in Linda is asynchronous, in that send operations are non-blocking. Our presentation of the syntax and the semantics of Linda follows those given in [55, 116].

In the light of its intuitive appeal and impact, Linda has received a fair amount of attention within the concurrency theory community. For instance, the relative expressive power of fragments of Linda has been studied in [55] and the paper [64] studies testing semantics, in the sense of De Nicola and Hennessy [63], over applicative and imperative process algebras that are inspired by Linda. The paper [63] also provides complete inequational axiomatizations of the studied calculi with respect to testing semantics.

Testing semantics can be viewed as the most natural notion of behavioural equivalence for a language from the programmer's perspective. Indeed, it is the formalization of the motto that 'two program fragments should be considered equivalent unless there is a context/test that tells them apart.' Testing semantics is, however, not very robust. In particular, if one extends a language with new features that increase the observational power of tests, the resulting notion of 'testing equivalence' for the extended language will be finer than the one for the original language. This means that the results one had worked hard to establish for the original language will have to be established anew.

Stateless bisimilarity [52, 62, 85, 116] is a variation on the classic notion of bisimilarity [106, 120] that is suitable for reasoning compositionally about open, concurrent and state-bearing systems. It is the finest notion of bisimilarity for state-bearing processes that one can find in the literature and comes equipped with a congruence rule format for operational rules [116]. It is therefore interesting to study its equational theory in the setting of a seminal language like Linda, not least because equational axiomatizations of stateless bisimilarity may form the core of axiom systems for coarser notions of equivalence over that language.

Contribution and structure The main contribution of this chapter is a ground-completeness result for stateless bisimilarity over Linda given in Section 2.3. We first present a complete axiom system for stateless bisimilarity over the sequential fragment of Linda without the *nask* primitive, which tests for the *absence* of a tuple in the tuple space (Theorem 2.3.1). Interestingly, it turns out that stateless bisimilarity over this fragment of Linda has the same axiomatization of standard bisimilarity, when the considered sub-language of Linda is viewed as Basic Process Algebra (BPA) with deadlock and the empty process [139]. We formalize the connection between the two languages and their respective semantics, culminating in Theorem 2.3.2.

Next we offer a ground-complete axiomatization of stateless bisimilarity over the full sequential fragment of Linda (Section 2.3.1). In this setting, we have to deal with the subtle interplay of *ask* and *nask* primitives, which test for the presence and absence of some tuple in the tuple space, respectively. In Theorem 2.3.3, we show that two equation schemas are enough to capture equationally the effect that combinations of *ask* and *nask* primitives may have on the behaviour of Linda terms.

Following rather standard lines, we give a ground-complete axiomatization of stateless bisimilarity over the full Linda language we consider in this chapter in Section 2.3.2.

We end the chapter with some concluding remarks and a suggestion for future research (Section 2.4).

2.2 Preliminaries

In this section we present the syntax and operational semantics for the classic, tuple-space-based coordination language Linda [56, 78]. (Our presentation follows those given in [55, 116].) Moreover, we introduce the notion of stateless bisimilarity and the basic definitions from equational logic used in this chapter.

We assume two signatures Σ_P and Σ_D for processes and data, respectively, which are sets of *function symbols* with fixed arities. A function symbol with arity zero is referred to as a *constant*. We let V_P and V_D denote two disjoint sets of, respectively, *process variables* and *data variables*. By $\mathbb{T}(\Sigma_P)$ and $\mathbb{T}(\Sigma_D)$ we denote the sets of open process and data terms, respectively, which are built using appropriate variables and function symbols by respecting their arities. Closed terms are terms without occurrences of variables. The sets of closed process and data term are $T(\Sigma_P)$ and $T(\Sigma_D)$, respectively. A substitution replaces a variable in an open term with some (possibly open) term. We call substitutions $\sigma : V_P \to \mathbb{T}(\Sigma_P)$ process substitutions and $\xi : V_D \to \mathbb{T}(\Sigma_D)$ data substitutions.

Linda's signature Σ_D for data (the so-called tuple space) consists of the constant $\emptyset$ for the empty tuple space, a (possibly infinite) set $\mathcal{U}$ of constants standing for memory tuples and a binary separator _ _ that is associative and commutative, but not idempotent, and has $\emptyset$ as left and right unit. (The store is a multiset of tuples.) The set $T(\Sigma_D)$ of closed *data terms* is given, therefore, by the following BNF grammar:

$$d ::= \emptyset \mid u \mid d\ d,$$

where $u \in \mathcal{U}$. Each data term d determines a multiset $\{u_1, \ldots, u_k\}$ of tuples in the obvious way. In what follows, we write $u \in d$ when there is at least one occurrence of the tuple u in the multiset denoted by d.

Following [55], the signature Σ_P for Linda is implicitly given by the BNF grammar defining the set $\mathbb{T}(\Sigma_P)$ of open *process terms* over a countably infinite set V_P of *process variables*:

$$t ::= x \mid \delta \mid \varepsilon \mid ask(u) \mid nask(u) \mid tell(u) \mid get(u) \mid t + t \mid t;t \mid t \parallel t,$$

where $x \in V_P$ and $u \in \mathcal{U}$. Closed terms are terms without occurrences of variables. The set of closed process terms is denoted by $T(\Sigma_P)$. A substitution σ is a function of type $V_P \to \mathbb{T}(\Sigma_P)$. A closed substitution is a substitution whose range in included in $T(\Sigma_P)$. We write $\sigma(t)$ for the term resulting by replacing each occurrence of a variable x in t with the term $\sigma(x)$. Note that $\sigma(t)$ is a closed term whenever σ is a closed substitution.

Intuitively, δ is a constant process that symbolizes deadlock, which satisfies no predicates and performs no actions. The constant ε denotes a process that satisfies the successful termination predicate, denoted by $\downarrow$ in what follows, and performs no action. The constants *ask*, *nask*, *tell*, and *get* are the basic Linda instructions for operating with the data component. $ask(u)$ and $nask(u)$ check whether tuple u is and, respectively, is not in the store. $tell(u)$ adds tuple u to the store, while $get(u)$ removes it if is present. The $ask(u)$, $get(u)$ and $nask(u)$ operations are blocking, in the sense that a process executing them blocks if u is not in the tuple space for *ask* and *get*, and if it is in the tuple space for *nask*. The operations $\_+\_$, $\_;\_$ and $\_\parallel\_$ are, respectively, the standard alternative, sequential and interleaving parallel composition operations familiar from process algebras—see, for instance, [25].

Definition 2.2.1 (Transition System Specification for Linda)**.** *The operational semantics of Linda is given in terms of a unary immediate termination predicate $\downarrow$ and a binary transition relation $\to$ over configurations of the form (p, d) with $p \in T(\Sigma_P)$ and $d \in T(\Sigma_D)$. Intuitively, $(p, d) \downarrow$ means that the process term p can terminate immediately in the context of the tuple space d, whereas*

$$(p, d) \to (p', d')$$

indicates that the configuration (p, d) can evolve into (p', d') in one computational step. Formally, $\downarrow$ and $\to$ are the least relations over configurations satisfying the following set of rules.

$$\frac{}{(\varepsilon, d)\downarrow} \qquad \frac{}{(ask(u), d\ u) \to (\varepsilon, d\ u)} \qquad \frac{}{(tell(u), d) \to (\varepsilon, d\ u)}$$

$$\frac{}{(get(u), d\ u) \to (\varepsilon, d)} \qquad \frac{}{(nask(u), d) \to (\varepsilon, d)}\ [u \notin d]$$

$$\frac{(x,d) \to (x',d')}{(x+y,d) \to (x',d')} \quad \frac{(y,d) \to (y',d')}{(x+y,d) \to (y',d')} \quad \frac{(x,d)\downarrow}{(x+y,d)\downarrow} \quad \frac{(y,d)\downarrow}{(x+y,d)\downarrow}$$

$$\frac{(x,d) \to (x',d')}{(x\ ;\ y,d) \to (x'\ ;\ y,d')} \quad \frac{(x,d)\downarrow \quad (y,d) \to (y',d')}{(x\ ;\ y,d) \to (y',d')} \quad \frac{(x,d)\downarrow \quad (y,d)\downarrow}{(x\ ;\ y,d)\downarrow}$$

$$\frac{(x,d) \to (x',d')}{(x \parallel y,d) \to (x' \parallel y,d')} \quad \frac{(y,d) \to (y',d')}{(x \parallel y,d) \to (x \parallel y',d')} \quad \frac{(x,d)\downarrow \quad (y,d)\downarrow}{(x \parallel y,d)\downarrow}$$

Note that the predicate $\downarrow$ is independent of the data component in a configuration, that is, if $(p,d)\downarrow$ for some $p \in T(\Sigma_P)$ and $d \in T(\Sigma_D)$, then $(p,d')\downarrow$ also holds for each $d' \in T(\Sigma_D)$.

Throughout the chapter we use the notion of *stateless bisimilarity* from [52, 62, 85, 116] as our notion of behavioural equivalence over closed Linda process terms. Stateless bisimilarity is the finest notion of bisimilarity for state-bearing processes that one can find in the literature. It is a variation on strong bisimilarity for processes with data in which the behaviour of process terms is compared in the context of all possible data terms, and that allows for interference from 'the environment' in the data part after each transition. This makes stateless bisimilarity suitable for reasoning compositionally about open concurrent systems.

Definition 2.2.2 (Stateless Bisimilarity)**.** *A relation $R \subseteq T(\Sigma_P) \times T(\Sigma_P)$ is a* stateless bisimulation *if, and only if, it is symmetric and the following conditions hold for each $(p,q) \in R$:*

- *for all $p' \in T(\Sigma_P)$ and $d, d' \in T(\Sigma_D)$, if $(p,d) \to (p',d')$ then there is some $q' \in T(\Sigma_P)$ such that $(q,d) \to (q',d')$ and $(p',q') \in R$;*
- *for each $d \in T(\Sigma_D)$, if $(p,d)\downarrow$ then $(q,d)\downarrow$.*

Two closed process terms p and q are stateless bisimilar*, denoted by $p \leftrightarrows_{\mathrm{sl}} q$, if there exists a stateless bisimulation R such that $(p,q) \in R$. Stateless bisimilarity is extended to open terms in the standard way: two open terms $t, t' \in \mathbb{T}(\Sigma_P)$ are stateless bisimilar when $\sigma(t) \leftrightarrows_{\mathrm{sl}} \sigma(t')$ holds for each closed substitution σ.*

Example 2.2.3. *The processes $tell(u) \parallel get(u)$ and $tell(u); get(u) + get(u); tell(u)$ are stateless bisimilar for each tuple u. Indeed, using the rules in Definition 2.2.1, it is not*

hard to check that the symmetric closure of the relation R, consisting of the pair

$$(tell(u) \parallel get(u), tell(u); get(u) + get(u); tell(u))$$

and the pairs

$$(\varepsilon \parallel get(u), \varepsilon; get(u)),\ (tell(u) \parallel \varepsilon, \varepsilon; tell(u)),\ (\varepsilon \parallel \varepsilon, \varepsilon),$$

is a stateless bisimulation.

Example 2.2.4. *Consider the terms $ask(u) + nask(u)$ and $ask(v) + nask(v)$, where u and v are (possibly different) tuples. By Definition 2.2.2, these terms are stateless bisimilar as they both transition to ε, independently of the data term $d \in T(\Sigma_D)$ they are paired up with, leaving the data term d unchanged.*

Definition 2.2.5 (Congruence)**.** *Let Σ be a signature. An equivalence relation $\sim$ over Σ-terms is a* congruence *if, for all $f \in \Sigma$ and closed terms $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$, where n is the arity of f, if $p_i \sim q_i$ for each $i \in \{1, \ldots, n\}$ then $f(p_1, \ldots, p_n) \sim f(q_1, \ldots, q_n)$.*

The following result is easy to show.

Proposition 2.2.6. *$\leftrightarrows_{\mathrm{sl}}$ is a congruence for Linda.*

Definition 2.2.7 (Axiom system, Derivability [25])**.** *An* axiom system *is a pair (Σ, E), where Σ is a signature and E is a set of* axioms (equations) *of the form $s = t$, where $s, t \in \mathbb{T}(\Sigma)$.*

By $\vdash$ we denote the well known notion of derivability *in equational logic—closure under substitutions and contexts, and the fact that equality is an equivalence relation are the means through which one can derive equations.*

An axiom system (Σ, E) is often identified with the set of equations E when the signature Σ is clear from the context.

Example 2.2.8. *The following axiom system E^1 over the signature for Linda coincides with the one for BPA with the empty process and δ proposed in [139, Table 4, page 291].*

$$
\begin{aligned}
x + y &= y + x && (e_1)\\
x + (y + z) &= (x + y) + z && (e_2)\\
\varepsilon + \varepsilon &= \varepsilon && (e_3)\\
\varepsilon + \delta &= \varepsilon && (e_4)\\
(x + y); z &= (x; z) + (y; z) && (e_5)\\
x; (y; z) &= (x; y); z && (e_6)\\
\delta; x &= \delta && (e_7)\\
\varepsilon; x &= x && (e_8)\\
x; \varepsilon &= x && (e_9)
\end{aligned}
$$

Using it, one can derive, for example, the following equations, which state that the + operation is idempotent and has δ as unit element:

$$x + x = x \tag{2.1}$$

$$x + \delta = x. \tag{2.2}$$

The basic sanity criterion for an axiom system is that it only allows one to derive valid equalities between terms. This is the so-called soundness property, which is formalized in the following definition in the setting of Linda modulo stateless bisimilarity.

Definition 2.2.9 (Soundness)**.** *An axiom system E over Σ_P is* sound *when, for all $s, t \in \mathbb{T}(\Sigma_P)$, if $E \vdash s = t$ then $s \leftrightarrows_{\mathrm{sl}} t$.*

The following result can be shown following standard lines.

Lemma 2.2.10. *E^1 is sound for stateless bisimilarity over Linda.*

Given a finite index set $I = \{i_1, \ldots, i_n\}$ and an indexed set of terms $\{t_i\}_{i \in I}$, we write $\sum_{i \in I} t_i$ for $t_{i_1} + \cdots + t_{i_n}$. An empty sum stands for δ. (The generalized sum notation is justified since, by the above lemma and the derivability of equation (2.2), the + operation is associative, commutative and has δ as unit element, modulo stateless bisimilarity.)

Ideally, one would like to have axiom systems that are strong enough to prove all the equalities that are valid with respect to the chosen notion of equivalence over Linda. As is customary in the literature on process calculi, in what follows we will focus on ground-complete axiom systems.

Definition 2.2.11 (Ground Completeness)**.** *An axiom system E is* ground complete *when, for all $p, q \in T(\Sigma_P)$, if $p \leftrightarrows_{\mathrm{sl}} q$ then $E \vdash p = q$.*

2.3 Axiomatization

In this section we provide a sound and ground complete axiomatization for stateless bisimilarity over the collection of Linda process terms. The axiom system is finite if the tuple names mentioned in the axioms are taken to be variables ranging over tuples. We build the axiomatization incrementally starting with all the operations except for *nask* and $\_ \| \_$. Then we discuss the issues *nask* poses and how to overcome them by means of two equations (Section 2.3.1). In order to avoid cluttering previous explanations, the axiomatization for $\_ \| \_$ comes only at the end and will be given following standard lines (Section 2.3.2).

We proceed by introducing the notion of *normal form* of a Linda process term, which plays a crucial role in proving the completeness of the proposed axiomatization.

Definition 2.3.1 (Normal Form)**.** *A term $t \in T(\Sigma_P)$ is in* normal form *if it is of the form $(\sum_{i\in I} a_i(u_i); t_i)[+\varepsilon]$ (that is, either $\sum_{i\in I} a_i(u_i); t_i$ or $(\sum_{i\in I} a_i(u_i); t_i) + \varepsilon$), with I a finite, possibly empty, index set, $a_i \in \{ask, nask, tell, get\}$, $u_i \in \mathcal{U}$ and t_i is in normal form for each $i \in I$.*

The terms $a_i(u_i); t_i$ $(i \in I)$ and ε, if present, are called the summands *of $(\sum_{i\in I} a_i(u_i); t_i)[+\varepsilon]$.*

An axiom system E over Σ_P is normalizing *for $t \in T(\Sigma_P)$ if there exists a term $t' \in T(\Sigma_P)$ in normal form such that $E \vdash t = t'$.*

In what follows, we let Σ_P^1 be Σ_P without the operations $\_ \| \_$ and $nask(u)$, for all $u \in U$, and Σ_P^2 be Σ_P without the operation $\_ \| \_$. The following lemma can be shown following standard lines.

Lemma 2.3.2. *E^1 is normalizing for each closed term in Σ_P^2.*

Theorem 2.3.1. *E^1 is sound and ground complete for stateless bisimilarity over $T(\Sigma_P^1)$.*

Proof. The soundness of the axiom system is given in Lemma 2.2.10.

In order to establish the ground completeness of E^1, we shall prove that, for all $p, q \in T(\Sigma_P^1)$,

$$p \leftrightarrows_{\mathrm{sl}} q \Rightarrow E^1 \vdash p = q.$$

To this end, we first define the function *height* that computes the height of the syntax tree associated with a term $t \in T(\Sigma_P^1)$:

$$height(p) = \begin{cases} 0 & \text{if } p \text{ is a constant,} \\ 1 + \max(height(p_1), height(p_2)) & \text{if } p = p_1 + p_2 \text{ or } p_1; p_2. \end{cases}$$

We prove the claim by induction on $M = \max(\mathit{height}(p), \mathit{height}(q))$.

Base case: If $M = 0$ then $p = q = \delta$, because none of the terms in the set $\{\varepsilon\} \cup \{\mathit{ask}(u), \mathit{tell}(u), \mathit{get}(u) \mid u \in \mathcal{U}\}$ is in normal form, and the claim follows immediately by reflexivity.

Inductive step, $M > 0$: In order to show that $p = q$ we argue that each summand of p is provably equal to a summand of q. We proceed by examining the possible forms a summand of p may have.

- Assume that ε is a summand of p. Then $(p, \emptyset) \downarrow$. As $p \leftrightarrows_{\mathrm{sl}} q$, it holds that $(q, \emptyset) \downarrow$. Since q is in normal form, ε is also a summand of q.
- Let $\mathit{ask}(u); p'$ be a summand of p. This yields that $(p, u) \to (\varepsilon; p', u)$. As $p \leftrightarrows_{\mathrm{sl}} q$ and q is in normal form, we have that $(q, u) \to (\varepsilon; q', u)$ for some q' such that $\varepsilon; p' \leftrightarrows_{\mathrm{sl}} \varepsilon; q'$. This means that q has the summand $\mathit{ask}(u); q'$. Indeed, the primitives $\mathit{tell}(u')$, for each $u' \in \mathcal{U}$, and $\mathit{get}(u)$ alter the tuple space u, and, if $u' \neq u$, neither $\mathit{ask}(u')$ nor $\mathit{get}(u')$ can be performed in the context of the tuple space u. Since $\max(\mathit{height}(p'), \mathit{height}(q')) < M$ and $p' \leftrightarrows_{\mathrm{sl}} q'$, we may use the induction hypothesis to infer that $E^1 \vdash p' = q'$. Hence, by substitutivity, $E^1 \vdash \mathit{ask}(u); p' = \mathit{ask}(u); q'$.
- Let $\mathit{tell}(u); p'$ be a summand of p. This yields that $(p, \emptyset) \to (\varepsilon; p', u)$. As $p \leftrightarrows_{\mathrm{sl}} q$ and q is in normal form, we have that $(q, \emptyset) \to (\varepsilon; q', u)$ for some q' such that $\varepsilon; p' \leftrightarrows_{\mathrm{sl}} \varepsilon; q'$. This means that q has the summand $\mathit{tell}(u); q'$, as $\mathit{tell}(u)$ is the only Linda primitive that can transform the empty tuple space into u. The proof now proceeds as in the above case.
- Let $\mathit{get}(u); p'$ be a summand of p. This yields that $(p, u) \to (\varepsilon; p', \emptyset)$. As $p \leftrightarrows_{\mathrm{sl}} q$ and q is in normal form, we have that $(q, u) \to (\varepsilon; q', \emptyset)$ for some q' such that $\varepsilon; p' \leftrightarrows_{\mathrm{sl}} \varepsilon; q'$. This means that q has the summand $\mathit{get}(u); q'$, as $\mathit{get}(u)$ is the only Linda primitive that can transform u into the empty tuple space. The proof now proceeds as above.

As each summand of p is provably equal to a summand of q, we have that $E^1 \vdash q = p + q$. (Note that, in the case that $p = \delta + \varepsilon$, this equality can be derived using equations (2.1) and (2.2).) By symmetry, $E^1 \vdash p = p + q$ too, and therefore $E^1 \vdash p = q$. □

The import of the above result is that stateless bisimilarity over $T(\Sigma_p^1)$ has the same axiomatization as standard strong bisimilarity [106, 120] over BPA with the

$$\frac{}{\varepsilon \downarrow} \qquad \frac{}{ask(u) \xrightarrow{ask(u)} \varepsilon} \qquad \frac{}{tell(u) \xrightarrow{tell(u)} \varepsilon} \qquad \frac{}{get(u) \xrightarrow{get(u)} \varepsilon}$$

$$\frac{x \xrightarrow{\alpha} x'}{x + y \xrightarrow{\alpha} x'} \qquad \frac{y \xrightarrow{\alpha} y'}{x + y \xrightarrow{\alpha} y'} \qquad \frac{x \downarrow}{x + y \downarrow} \qquad \frac{y \downarrow}{x + y \downarrow}$$

$$\frac{x \xrightarrow{\alpha} x'}{x \,;\, y \xrightarrow{\alpha} x' \,;\, y} \qquad \frac{x \downarrow \quad y \xrightarrow{\alpha} y'}{x \,;\, y \xrightarrow{\alpha} y'} \qquad \frac{x \downarrow \quad y \downarrow}{x \,;\, y \downarrow}$$

Figure 2.1: SOS rule for the labelled transition system semantics for $T(\Sigma_p^1)$ ($\alpha \in \mathcal{A}^+$)

empty process and δ. This may seem surprising at first sight, since the definition of stateless bisimilarity over Linda given in Definition 2.2.2 is based on an unlabelled transition system semantics, whereas strong bisimilarity is based on a labelled transition system semantics. However, as the proof of the ground-completeness result given above indicates, the effect on the tuple space of the execution of the primitive operations in Linda considered so far, in combination with the definition of stateless bisimilarity, essentially encodes the primitive operation that is being executed in an unlabelled computational step. We now make this intuition precise, by showing how the problem of axiomatizing stateless bisimilarity over $T(\Sigma_p^1)$ can be reduced to that of axiomatizing ordinary bisimilarity over that language.

Let $\mathcal{A}^+ = \{ask(u), tell(u), get(u) \mid u \in \mathcal{U}\}$. We define a partial function upd : $\mathcal{A}^+ \times T(\Sigma_D) \to T(\Sigma_D)$ as follows, where $d \in T(\Sigma_D)$ and $u \in \mathcal{U}$:

$$\begin{aligned} \mathrm{upd}(ask(u), d\ u) &= d\ u, \\ \mathrm{upd}(get(u), d\ u) &= d \qquad \text{and} \\ \mathrm{upd}(tell(u), d) &= d\ u. \end{aligned}$$

Intuitively, $\mathrm{upd}(\alpha, d) = d'$ holds for some $\alpha \in \mathcal{A}^+$ and $d, d' \in T(\Sigma_D)$ if, and only if, the primitive operation α can be executed in the context of the tuple space represented by d and its execution results in the tuple space represented by d'. For example, the first equation in the definition of the function upd specifies that $\mathrm{upd}(ask(u), d)$ is only defined if $u \in d$, and that the execution of $ask(u)$ leaves d unchanged.

The following lemma connects the transition system semantics for $T(\Sigma_p^1)$ given in Definition 2.2.1 with the standard labelled transition system semantics that $T(\Sigma_p^1)$ inherits when viewed as BPA, with ε and δ, over the set of actions $\mathcal{A}^+$, which is given in Figure 2.1.

Lemma 2.3.3. *For all $p, p' \in T(\Sigma_P^1)$ and $d, d' \in T(\Sigma_D)$,*

$$(p, d) \to (p', d') \Leftrightarrow \exists \alpha \in \mathcal{A}^+.\ p \xrightarrow{\alpha} p' \text{ and } \mathrm{upd}(\alpha, d) = d'.$$

Proof. Both implications can be shown by induction on the proof of the relevant transition. We omit the straightforward details. □

As an easy, but useful, corollary of the above lemma and of the definition of the upd function, we have the following observations.

Corollary 2.3.4. *For all $p, p' \in T(\Sigma_P^1)$ and $u \in \mathcal{U}$, the following statements hold:*

1. *$(p, u) \to (p', u)$ if, and only if, $p \xrightarrow{ask(u)} p'$;*
2. *$(p, u) \to (p', \emptyset)$ if, and only if, $p \xrightarrow{get(u)} p'$; and*
3. *$(p, \emptyset) \to (p', u)$ if, and only if, $p \xrightarrow{tell(u)} p'$.*

For the sake of clarity, we now recall the standard definition of bisimilarity in the presence of the termination predicate.

Definition 2.3.5 (Bisimilarity)**.** *A relation $R \subseteq T(\Sigma_P^1) \times T(\Sigma_P^1)$ is a* bisimulation *if and only if it is symmetric and, whenever $(p, q) \in R$, the following conditions hold:*

- $\forall p' \in T(\Sigma_P^1).\ p \xrightarrow{\alpha} p' \Rightarrow \exists q' \in T(\Sigma_P^1).\ q \xrightarrow{\alpha} q' \wedge (p', q') \in R$;
- $p \downarrow \Rightarrow q \downarrow$.

Two closed process terms p and q are bisimilar*, denoted by $p \leftrightarrows q$, if there exists a bisimulation R such that $(p, q) \in R$. Bisimilarity is extended to open terms in the standard way: two open terms $t, t' \in \mathbb{T}(\Sigma_P^1)$ are bisimilar when $\sigma(t) \leftrightarrows \sigma(t')$ holds for each closed substitution σ.*

Theorem 2.3.2. *Stateless bisimilarity and bisimilarity coincide over $T(\Sigma_P^1)$—that is, $p \leftrightarrows_{\mathrm{sl}} q$ if, and only if, $p \leftrightarrows q$, for all $p, p' \in T(\Sigma_P^1)$.*

Proof. Using Corollary 2.3.4, it is not hard to show that $\leftrightarrows_{\mathrm{sl}}$ is a bisimulation and, using Lemma 2.3.3, one proves that $\leftrightarrows$ is a stateless bisimulation. In establishing both claims, we use the simple observation that

$$p \downarrow \text{ if, and only if, } (p, d) \downarrow \text{ for all } d \in T(\Sigma_D),$$

for all $p \in T(\Sigma_P^1)$, for all $p \in T(\Sigma_P^1)$, and that, as remarked earlier, the predicate $\downarrow$ from Definition 2.2.1 is independent of the data component in a configuration. □

The above result yields an alternative, but less direct, proof of Theorem 2.3.1 by reducing the problem of axiomatizing stateless bisimilarity over $T(\Sigma_P^1)$ to that of axiomatizing bisimilarity over that language. It is well known that bisimilarity is axiomatized by the axiom system E^1 over $T(\Sigma_P^1)$—see, e.g., [32].

2.3.1 Adding the *nask* operations

As we proved above, the axiom system E^1 is ground complete for stateless bisimilarity over $T(\Sigma_P^1)$. However, when we add the *nask* operations to Σ_P^1, E^1 is *not* ground complete any more. To see this, recall that in Example 2.2.4 we argued that $ask(u) + nask(u) \leftrightarrows_{\mathrm{sl}} ask(v) + nask(v)$ holds, even when the tuples u and v are distinct. The axiom system E^1, however, does not suffice to prove that equality when $u \neq v$.

Consider the axiom system E^2, which is obtained by adding the following equations to E^1.

$$ask(u) + nask(u) + ask(v) = ask(u) + nask(u) \text{ for all } u, v \in \mathcal{U} \quad (e_{10})$$
$$ask(u) + nask(u) + nask(v) = ask(u) + nask(u) \text{ for all } u, v \in \mathcal{U} \quad (e_{11})$$

Note that, using the above equations, one may derive the following ones:

$$ask(u) + nask(u) + ask(w) = ask(v) + nask(v) \text{ for all } u, v, w \in \mathcal{U},$$
$$ask(u) + nask(u) + nask(w) = ask(v) + nask(v) \text{ for all } u, v, w \in \mathcal{U}.$$

Note that, due to the unlabelled nature of the transition relation, all "one-step" processes that never block and that do not change the data component are equivalent.

Our order of business in the remainder of this subsection is to show that E^2 is sound and ground complete modulo stateless bisimilarity over $T(\Sigma_P^2)$. (Recall that Σ_P^2 is Σ_P without the operation $\_\|\_$.)

For each $p = (\sum_{i\in I} a_i(u_i); p_i)[+\varepsilon] \in T(\Sigma_P^2)$ in normal form, let:

- $p_{an} = \sum_{i\in I,\ a_i \text{ is } ask \text{ or } nask} a_i(u_i); p_i$,
- $p_{gt} = \sum_{i\in I,\ a_i \text{ is } get \text{ or } tell} a_i(u_i); p_i$.

The following 'decomposition lemma' will be useful in establishing the desired completeness result.

Lemma 2.3.6. *Let $p, q \in T(\Sigma_p^2)$ be two terms in normal form. Then p and q are stateless bisimilar if, and only if,*

1. $p_{an} \leftrightarrows_{\mathrm{sl}} q_{an}$,
2. $p_{gt} \leftrightarrows_{\mathrm{sl}} q_{gt}$, *and*
3. ε *is a summand of p if, and only if, it is a summand of q.*

Proof. The 'if' implication follows because, by Proposition 2.2.6, stateless bisimilarity is preserved by $+$. Assume now that p and q are stateless bisimilar normal forms. We shall show that statements 1–3 hold. Statement 3 is immediate since, for each term t in normal form, $(t, \emptyset) \downarrow$ if, and only if, ε is a summand of t. To prove the other two statements, it suffices to show that the relation

$$R = \{(p_{an}, q_{an}), (p_{gt}, q_{gt}) \mid p, q \text{ are in normal form and } p \leftrightarrows_{\mathrm{sl}} q\} \cup \leftrightarrows_{\mathrm{sl}}$$

is a stateless bisimulation. First of all, note that R is symmetric since so is $\leftrightarrows_{\mathrm{sl}}$. We limit ourselves to presenting the verification of the stateless bisimulation conditions for a pair of the form $(p_{an}, q_{an}) \in R$. The proof for pairs of the form $(p_{gt}, q_{gt}) \in R$ is similar and we omit it.

Let $d \in T(\Sigma_D)$. By definition of p_{an}, we have that $(p_{an}, d) \downarrow$ does not hold. Hence the second clause in Definition 2.2.2 is met vacuously. Assume now that $(p_{an}, d) \to (p', d')$ for some p' and d'. Then there is a summand $a_i(u_i); p_i$ of p_{an} such that '$a_i(u_i)$ is enabled in the context of d', $p' = \varepsilon; p_i$ and $d = d'$. Since $a_i(u_i); p_i$ is also a summand of p, we have that $(p, d) \to (p', d)$. From our assumption that p and q are stateless bisimilar, we infer that $(q, d) \to (q', d)$ for some q' such that $p' \leftrightarrows_{\mathrm{sl}} q'$. Since the execution of *get* and *tell* primitives modifies the tuple space d, the above transition is due to a summand of q that is also a summand of q_{an}. Thus, $(q_{an}, d) \to (q', d)$ for some q' such that $p' \leftrightarrows_{\mathrm{sl}} q'$. Since $\leftrightarrows_{\mathrm{sl}}$ is included in R, we are done. □

Theorem 2.3.3. *E^2 is sound and ground complete modulo stateless bisimilarity over $T(\Sigma_p^2)$.*

Proof. It is easy to check that equations (e_{10}) and (e_{11}) are sound.

In order to establish the ground completeness of E^2, we shall prove that, for all $p, q \in T(\Sigma_p^2)$,

$$p \leftrightarrows_{\mathrm{sl}} q \Rightarrow E^2 \vdash p = q.$$

Assume that $p \leftrightarrows_{\mathrm{sl}} q$. We shall prove the claim above by induction on $M = \max(\mathit{height}(p), \mathit{height}(q))$, where the function *height* from the proof of Theorem 2.3.1 is extended to $T(\Sigma_p^2)$ by setting

$$\mathit{height}(\mathit{nask}(u)) = 0.$$

By Lemma 2.3.2, we may assume that p, q are in normal form. Our induction hypothesis is that $E^2 \vdash p' = q'$ for all p', q' such that $p' \leftrightarrows_{\mathrm{sl}} q'$ and $M > \max(\mathit{height}(p'), \mathit{height}(q'))$.

By Lemma 2.3.6, p_{gt} and q_{gt} are stateless bisimilar and they can be proved equal by mimicking the proof of Theorem 2.3.1. Using again Lemma 2.3.6, we have that p_{an} and q_{an} are also stateless bisimilar. We claim that

$$E^2 \vdash p_{an} = q_{an}, \tag{2.3}$$

from which the equality $p = q$ follows by substitutivity, also in the light of the last statement in Lemma 2.3.6.

In order to show the above claim, note, first of all, that, modulo E^2, we may assume that p_{an} and q_{an} are in the adapted normal forms

$$\begin{aligned} p_{an} &= \sum_{i \in I} ((\sum_{j \in J_i} a_j(u_j)))\,; p_i \quad \text{and} \\ q_{an} &= \sum_{i \in I} ((\sum_{k \in K_i} b_k(v_k)))\,; p_i, \end{aligned}$$

where J_i and K_i are non-empty index sets ($i \in I$), $a_j, b_k \in \{\mathit{ask}, \mathit{nask}\}$ and $E^2 \nvdash p_i = p_j$ (or, equivalently, $p_i \not\leftrightarrows_{\mathrm{sl}} p_j$), for all $i, j \in I$ such that $i \neq j$. To see this, assume that $p_{an} = \sum_{\ell \in L} a_\ell(u_\ell)\,; p_\ell$ and that $p_{\ell_1} \leftrightarrows_{\mathrm{sl}} p_{\ell_2}$, for some $\ell_1, \ell_2 \in L$. Since

$$\max(\mathit{height}(p_{\ell_1}), \mathit{height}(p_{\ell_2})) < \mathit{height}(p_{an}) \leq \mathit{height}(p) \leq M,$$

we may use the inductive hypothesis to obtain that $E^2 \vdash p_{\ell_1} = p_{\ell_2}$. Therefore, modulo E^2,

$$p_{an} = \sum_{\ell \in L} a_\ell(u_\ell)\,; p'_\ell,$$

where $p'_\ell \in \{p_h \mid h \in L \text{ and } p_\ell \leftrightarrows_{\mathrm{sl}} p_h\}$ is a canonical representative of the equivalence class of the 'suffixes' of p_{an} that are stateless bisimilar to p_ℓ, for each $\ell \in L$. For this reason, we can group all the summands of p_{an} with the same suffix p_i modulo E^2 by applying equation (e_5) form right to left as needed. This puts p_{an} in the desired

form, namely

$$p_{an} = \sum_{i \in I}((\sum_{j \in J_i} a_j(u_j)));p_i.$$

(Note that each J_i is non-empty.) Let $q_{an} = \sum_{h \in H} b_h(v_h);q_h$. Since p_{an} and q_{an} are stateless bisimilar, it is not hard to see that, for each $h \in H$, there is some $i \in I$ such that $p_i \leftrightarrows_{\mathrm{sl}} q_h$. Using the inductive hypothesis as above, the equality $p_i = q_h$ can be proved from E^2. This means that q_{an} can be put in the form $\sum_{i \in I}((\sum_{k \in K_i} b_k(v_k)));p_i$, by substitutivity and applying equation (e_5) form right to left as needed. (Note that, since $p_{an} \leftrightarrows_{\mathrm{sl}} q_{an}$, each K_i is non-empty.)

In order to show claim (2.3), it therefore suffices to show that $C_i = \sum_{j \in J_i} a_j(u_j)$ is provably equal to $D_i = \sum_{k \in K_i} b_k(v_k)$, for each $i \in I$.

For a fixed $i \in I$, we consider the two sums of *ask* and *nask* terms C_i and D_i. Since E^2 proves equation (2.1), to the effect that + is idempotent, we may assume in what follows that all the summands of C_i and D_i are different.

We shall prove that $E^2 \vdash C_i = D_i$ by case analysis on their possible form.

1. $C_i = ask(u) + nask(u) + C'$ and $D_i = ask(v) + nask(v) + D'$, for some $u, v \in \mathcal{U}$ and C', D'. (Note that, in the light of equation (2.2), C' and D' may be δ.) Then the equality $C_i = D_i$ can be shown by using equations (e_{10}) and (e_{11}) repeatedly.

2. $C_i = ask(u) + nask(u) + C$, for some $u \in \mathcal{U}$ and C, and $D_i = \sum_{k \in K_i} b_k(v_k)$, with $v_{k_1} \neq v_{k_2}$ whenever $k_1 \neq k_2$, for all $k_1, k_2 \in K_i$. We shall argue that this case is impossible, since it contradicts the assumption that p_{an} and q_{an} are stateless bisimilar.

   By using equations (e_{10}) and (e_{11}) repeatedly, we can derive the equation $C_i = ask(u) + nask(u)$. It is easy to see that $(C_i, d) \to (\varepsilon, d)$, for each $d \in T(\Sigma_D)$. Therefore $(p_{an}, d) \to (\varepsilon; p_i, d)$, for each $d \in T(\Sigma_D)$. However, since $D_i = \sum_{k_1 \in K_i^1} ask(v_{k_1}) + \sum_{k_2 \in K_i^2} nask(v_{k_2})$ with $K_i^1 \cap K_i^2 = \emptyset$ and $K_i^1 \cup K_i^2 = K_i$, the tuple space $d' = \{v_{k_2} \mid k_2 \in K_2\}$ 'blocks' D_i (no transition can be performed from (D_i, d')). This means that (q_{an}, d') does not have a transition leading to $(\varepsilon; p_i, d')$, which contradicts the assumption that p_{an} and q_{an} are stateless bisimilar.

3. $D_i = ask(u) + nask(u) + D$, for some $u \in \mathcal{U}$ and D, and $C_i = \sum_{k \in K_i} b_k(v_k)$, with $v_{k_1} \neq v_{k_2}$ whenever $k_1 \neq k_2$, for all $k_1, k_2 \in K_i$. This case is symmetric to the one above.

4. Assume that none of the previous cases applies. Let

$$C_i = \sum_{j_1 \in J_i^1} ask(v_{j_1}) + \sum_{j_2 \in J_i^2} nask(v_{j_2}),$$

with $J_i^1 \cap J_i^2 = \emptyset$ and $J_i^1 \cup J_i^2 = J_i$, and

$$D_i = \sum_{k_1 \in K_i^1} ask(v_{k_1}) + \sum_{k_2 \in K_i^2} nask(v_{k_2}),$$

with $K_i^1 \cap K_i^2 = \emptyset$ and $K_i^1 \cup K_i^2 = K_i$. We have that $v_{j_1} \neq v_{j_2}$, for each $j_1 \in J_i^1$ and $j_2 \in J_i^2$, and $v_{k_1} \neq v_{k_2}$, for each $k_1 \in K_i^1$ and $k_2 \in K_i^2$.

We show that each summand in C_i appears in D_i, and vice versa, by reductio ad absurdum. We consider the following four cases, each of which contradicts the assumption that p_{an} and q_{an} are stateless bisimilar.

- Suppose $J_i^1 \setminus K_i^1 \neq \emptyset$. Then $d = \{v_l \mid l \in (J_i^1 \setminus K_i^1) \cup K_i^2\}$ 'blocks' D_i, but $(C_i, d) \rightarrow (\varepsilon, d)$. This means that (q_{an}, d) does not have a transition leading to $(\varepsilon; p_i, d)$, whereas (p_{an}, d) does. This contradicts the assumption that p_{an} and q_{an} are stateless bisimilar.
- Suppose $K_i^1 \setminus J_i^1 \neq \emptyset$. The proof for this case is similar to that for the previous one.
- Suppose $J_i^2 \setminus K_i^2 \neq \emptyset$. Then $d = \{v_l \mid l \in K_i^2\}$ 'blocks' D_i, but $(C_i, d) \rightarrow (\varepsilon, d)$. This means that (q_{an}, d) does not have a transition leading to $(\varepsilon; p_i, d)$, whereas (p_{an}, d) does. This contradicts the assumption that p_{an} and q_{an} are stateless bisimilar.
- Suppose $K_i^2 \setminus J_i^2 \neq \emptyset$. The proof for this case is similar to that for the previous one.

Concluding, $E^2 \vdash C_i = D_i$ for each $i \in I$. This means that $E^2 \vdash p_{an} = q_{an}$, and we are done. □

2.3.2 Adding parallel composition

Our goal in this section is to axiomatize stateless bisimilarity over the full Linda language studied in this chapter. Consider the signature Σ_P^3, an extension of Σ_P with the binary left merge operation $\_ \between \_$, which stems from [43], defined by the rules:

$$\frac{(x,d) \rightarrow (x',d')}{(x \between y,d) \rightarrow (x' \parallel y,d')} \qquad \frac{(x,d)\downarrow \quad (y,d)\downarrow}{(x \between y,d)\downarrow}$$

As is well known (see, e.g., [107]), the left merge operation is necessary in order to obtain finite equational axiomatizations of bisimilarity in process algebras.

Consider the axiom system E^3 which is E^2 enriched with the following equations.

$$\begin{aligned}
x \parallel y &= x \between y + y \between x && (e_{12})\\
(x+y) \between z &= (x \between z) + (y \between z) && (e_{13})\\
(a(u);x) \between y &= a(u);(x \parallel y) \text{ for all } a \in \{ask, nask, tell, get\} \text{ and } u \in \mathcal{U} && (e_{14})\\
\varepsilon \between (x+y) &= \varepsilon \between x + \varepsilon \between y && (e_{15})\\
\varepsilon \between (a(u);y) &= \delta \text{ for all } a \in \{ask, nask, tell, get\} \text{ and } u \in \mathcal{U} && (e_{16})\\
\varepsilon \between \varepsilon &= \varepsilon && (e_{17})\\
\varepsilon \between \delta &= \delta && (e_{18})\\
\delta \between x &= \delta && (e_{19})
\end{aligned}$$

Theorem 2.3.4. *E^3 is sound and ground complete modulo stateless bisimilarity over $T(\Sigma_P^3)$.*

Proof. The soundness of E^3 modulo stateless bisimilarity can be shown following standard lines. The ground completeness of E^3 modulo stateless bisimilarity can be reduced to that of E^2 over $T(\Sigma_P^2)$. Indeed, by induction on the size of terms, one can show that the equations $(e_{12} - e_{19})$ can be used to eliminate each occurrence of $\parallel$ and $\between$ from terms. □

2.4 Conclusions

In this chapter, we have presented a sound and ground complete axiomatization for stateless bisimilarity over the collection of Linda process terms. The axiom system is finite if the tuple names mentioned in the axioms are taken to be variables ranging over tuples.

The axiom systems we present in this chapter may form the core of axiom systems for coarser notions of equivalence over Linda. In particular, an interesting direction for future research is the development of a complete axiomatization for the notion of behavioural equivalence over Linda studied in [55].

Chapter 3

Axiomatizing GSOS with Predicates

3.1 Introduction

One of the greatest challenges in computer science is the development of rigorous methods for the specification and verification of reactive systems, *i.e.*, systems that compute by interacting with their environment. Typical examples include embedded systems, control programs and distributed communication protocols. Over the last three decades, process algebras, such as ACP [28], CCS [106] and CSP [95], have been successfully used as common languages for the description of both actual systems and their specifications. In this context, verifying whether the implementation of a reactive system complies to its specification reduces to proving that the corresponding process terms are related by some notion of behavioural equivalence or preorder [79].

One approach to proving equivalence between two terms is to exploit the equational style of reasoning supported by process algebras. In this approach, one obtains a (ground-)complete axiomatization of the behavioural relation of interest and uses it to prove the equivalence between the terms describing the specification and the implementation by means of equational reasoning, possibly in conjunction with proof rules to handle recursively-defined process specifications.

Finding a "finitely specified", (ground-)complete axiomatization of a behavioural equivalence over a process algebra is often a highly non-trivial task. However, as shown in [6] in the setting of bisimilarity [106, 120], this process can be automated

for process languages with an operational semantics given in terms of rules in the GSOS format of Bloom, Istrail and Meyer [49]. In that reference, Aceto, Bloom and Vaandrager provided an algorithm that, given a GSOS language as input, produces as output a "conservative extension" of the original language with auxiliary operators together with a finite axiom system that is sound and ground-complete with respect to bisimilarity (see, *e.g.*, [2, 76, 136] for further results in this line of research). As the operational specification of several operators often requires a clear distinction between successful termination and deadlock, an extension of the above-mentioned approach to the setting of GSOS with a predicate for termination was proposed in [35].

Contribution In this chapter we contribute to the line of the work in [6] and [35]. Inspired by [35], we introduce the *preg* rule format, a natural extension of the GSOS format with an arbitrary collection of predicates such as termination, convergence and divergence. We further adapt the theory in [6] to this setting and give a procedure for obtaining ground-complete axiomatizations for bisimilarity over *preg* systems. More specifically, we develop a general procedure that, given a *preg* language as input, automatically synthesizes a conservative extension of that language and a finite axiom system that, in conjunction with an infinitary proof rule, yields a sound and ground-complete axiomatization of bisimilarity over the extended language. The work we present in this chapter is based on the one reported in [6, 35]. However, handling more general predicates than immediate termination requires the introduction of some novel technical ideas. In particular, the problem of axiomatizing bisimilarity over a *preg* language is reduced to that of axiomatizing that relation over finite trees whose nodes may be labelled with predicates. In order to do so, one needs to take special care in axiomatizing negative premises in rules that may have positive and negative premises involving predicates and transitions.

The results of the current chapter have been used for the implementation of a Maude [59] tool, presented in Chapter 7, that enables the user to specify *preg* systems in a uniform fashion, and that automatically derives the associated axiomatizations. The tool is available at `http://goriac.info/tools/preg-axiomatizer/`. This paves the way to checking bisimilarity over process terms by means of theorem-proving techniques for a large class of systems that can be expressed using *preg* language specifications.

Structure In Section 3.2 we introduce the *preg* rule format. In Section 3.3 we introduce an appropriate "core" language for expressing finite trees with predicates. We also provide a ground-complete axiomatization for bisimilarity over this type of trees, as our aim is to prove the completeness of our final axiomatization by head normalizing general *preg* terms, and therefore by reducing the completeness problem for arbitrary languages to that for trees.

Head normalizing general *preg* terms is not a straightforward process. Therefore, following [6], in Section 3.4 we introduce the notion of smooth and distinctive operation, adapted to the current setting. These operations are designed to "capture the behaviour of general *preg* operations", and are defined by rules satisfying a series of syntactic constraints with the purpose of enabling the construction of head normalizing axiomatizations. Such axiomatizations are based on a collection of equations that describe the interplay between smooth and distinctive operations, and the operations in the signature for finite trees. The existence of a sound and ground-complete axiomatization characterizing the bisimilarity of *preg* processes is finally proven in Section 3.5. A technical discussion on why it is important to handle predicates as first class notions, instead of encoding them by means of transition relations, is presented in Section 3.6. In Section 3.7 we draw some conclusions and provide pointers to future work.

This chapter is an extended version of [7]. It offers the proofs of results that were announced without proof in this reference, as well as a new theory, in Appendix 3.I, on many different types of predicates a system could include, and how to axiomatize these systems.

3.2 GSOS with predicates

In this section we present the *preg* systems which are a generalization of GSOS [49] systems.

Consider a countably infinite set V of *process variables* (usually denoted by x, y, z) and a signature Σ consisting of a set of *operations* (denoted by f, g). The set of *process terms* $\mathbb{T}(\Sigma)$ is inductively defined as follows: each variable $x \in V$ is a term; if $f \in \Sigma$ is an operation of arity l, and if $S_1, \ldots, S_l$ are terms, then $f(S_1, \ldots, S_l)$ is a term. We write $T(\Sigma)$ in order to represent the set of *closed process terms* (*i.e.*, terms that do not contain variables), ranged over by t, s. A *substitution* σ is a function of type $V \to \mathbb{T}(\Sigma)$. If the range of a substitution is included in $T(\Sigma)$, we say that

it is a *closed substitution*. Moreover, we write $[x \mapsto t]$ to represent a substitution that maps the variable x to the term t. Let $\vec{x} = x_1, \ldots, x_n$ be a sequence of pairwise distinct variables. A Σ-*context* $C[\vec{x}]$ is a term in which at most the variables $\vec{x}$ appear. For instance, $f(x, f(x, c))$ is a Σ-context, if the binary operation f and the constant c are in Σ.

Let $\mathcal{A}$ be a finite, nonempty set of *actions* (denoted by a, b, c). A *positive transition formula* is a triple (S, a, S') written $S \xrightarrow{a} S'$, with the intended meaning: process S performs action a and becomes process S'. A *negative transition formula* (S, a) written $S \overset{a}{\nrightarrow}$, states that process S cannot perform action a. Note that S, S' may contain variables. The "intended meaning" applies to closed process terms.

We now define *preg* – *pr*edicate *e*xtension of the *G*SOS rule format. Let $\mathcal{P}$ be a finite set of *predicates* (denoted by P, Q). A *positive predicate formula* is a pair (P, S), written PS, saying that process S satisfies predicate P. Dually, a *negative predicate formula* $\neg P\, S$ states that process S does not satisfy predicate P.

Definition 3.2.1 (*preg* rule format)**.** *Consider $\mathcal{A}$, a set of actions, and $\mathcal{P}$, a set of predicates.*

1. *A preg* transition rule *for an l-ary operation f is a deduction rule of the form:*

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+, j \in I_i^+\} \quad \{P_{ij}x_i \mid i \in J^+, j \in J_i^+\} \quad \{x_i \overset{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

   where

   (a) *$x_1, \ldots, x_l$ and y_{ij} $(i \in I^+, j \in J^+)$ are pairwise distinct variables;*

   (b) *$I^+, J^+, I^-, J^- \subseteq L = \{1, \ldots, l\}$ and each I_i^+ and J_i^+ is finite;*

   (c) *a_{ij}, b and c are actions in $\mathcal{A}$ $(\mathcal{B}_i \subseteq \mathcal{A})$; and*

   (d) *P_{ij} and Q are predicates in $\mathcal{P}$ $(\mathcal{Q}_i \subseteq \mathcal{P})$.*

2. *A preg* predicate rule *for an l-ary operation f is a deduction rule similar to the one above, with the only difference that its conclusion has the form $P(f(x_1, \ldots, x_l))$ for some $P \in \mathcal{P}$.*

Let ρ be a *preg* (transition or predicate) rule for f. The symbol f is the *principal operation* of ρ. All the formulas above the line are *antecedents* and the formula below is the *consequent*. We say that a position i for ρ is *tested positively* if $i \in I^+ \cup J^+$

and $I_i^+ \cup J_i^+ \neq \emptyset$. Similarly, i is *tested negatively* if $i \in I^- \cup J^-$ and $\mathcal{B}_i \cup \mathcal{Q}_i \neq \emptyset$. Whenever ρ is a transition rule for f, we say that $f(\vec{x})$ is the *source*, $C[\vec{x}, \vec{y}]$ is the *target*, and c is the *action* of ρ. Whenever ρ is a predicate rule for f, we call $f(\vec{x})$ the *test* of ρ.

In order to avoid confusion, if in a certain context we use more than one rule, e.g. ρ, ρ', we parameterize the corresponding sets of indices with the name of the rule, *e.g.*, I_ρ^+, $J_{\rho'}^-$.

Definition 3.2.2 (*preg* system)**.** *A* preg *system is a pair $G = (\Sigma_G, \mathcal{R}_G)$, where Σ_G is a finite signature and $\mathcal{R}_G = \mathcal{R}_G^{\mathcal{A}} \cup \mathcal{R}_G^{\mathcal{P}}$ is a finite set of* preg *rules over Σ_G ($\mathcal{R}_G^{\mathcal{A}}$ and $\mathcal{R}_G^{\mathcal{P}}$ represent the transition and, respectively, the predicate rules of G).*

Definition 3.2.3 (Transition relation)**.** *A* transition relation *over a signature Σ is a relation $\rightsquigarrow \subseteq T(\Sigma) \times \mathcal{A} \times T(\Sigma)$. We write $t \overset{a}{\rightsquigarrow} t'$ as an abbreviation for $(t, a, t') \in \rightsquigarrow$.*

Definition 3.2.4 (Predicate relation)**.** *A* predicate relation *over a signature Σ is a relation $\propto \subseteq \mathcal{P} \times T(\Sigma)$. We write Pt as an abbreviation for $(P, t) \in \propto$.*

Definition 3.2.5 (Σ-substitution)**.** *A* (closed) Σ-substitution *is a function σ from variables to (closed) terms over the signature Σ. For a term S, we write $S\sigma$ for the result of substituting $\sigma(x)$ for each x in S. For $\vec{S}$ and $\vec{x}$ of the same length, $\langle \vec{S}/\vec{x} \rangle$ represents the substitution that replaces the i-th variable of $\vec{x}$ by the i-th term of $\vec{S}$, and is the identity function on all the other variables.*

Definition 3.2.6 (Rule satisfiability)**.** *Consider $\rightsquigarrow$ a transition relation, $\propto$ a predicate relation, and σ a closed substitution. For each (transition, resp. predicate) formula γ, the predicate $\rightsquigarrow, \propto, \sigma \models \gamma$ is defined as follows:*

$$\begin{aligned}
\rightsquigarrow, \propto, \sigma \models S \overset{a}{\rightarrow} S' &\triangleq (S\sigma, a, S'\sigma) \in \rightsquigarrow, \\
\rightsquigarrow, \propto, \sigma \models S \overset{a}{\not\rightarrow} &\triangleq \nexists t \,.\, (S\sigma, a, t) \in \rightsquigarrow, \\
\rightsquigarrow, \propto, \sigma \models PS &\triangleq (P, S\sigma) \in \propto, \\
\rightsquigarrow, \propto, \sigma \models \neg PS &\triangleq (P, S\sigma) \notin \propto.
\end{aligned}$$

Let H be a set of (transition and/or predicate) formulas. We define

$$\rightsquigarrow, \propto, \sigma \models H \triangleq (\forall \gamma \in H) \,.\, \rightsquigarrow, \propto, \sigma \models \gamma$$

Let $\rho = \dfrac{H}{\gamma}$ be a preg *rule. We define*

$$\rightsquigarrow, \propto, \sigma \models \rho \triangleq (\rightsquigarrow, \propto, \sigma \models H) \Rightarrow (\rightsquigarrow, \propto, \sigma \models \gamma)$$

Definition 3.2.7 (Sound and supported relations)**.** *Consider $G = (\Sigma_G, \mathcal{R}_G)$ a* preg *system, $\rightsquigarrow$ a transition relation over Σ_G, and $\propto$ a predicate relation over Σ_G. Then $\rightsquigarrow$ and $\propto$ are* sound *for G if, and only if, for every rule $\rho \in \mathcal{R}_G$ and every closed substitution σ, we have $\rightsquigarrow, \propto, \sigma \models \rho$.*
A transition formula $t \overset{a}{\rightsquigarrow} t'$ (resp. a predicate formula Pt) is supported *by some rule $\rho = \dfrac{H}{\gamma} \in \mathcal{R}_G$ if, and only if, there exists a substitution σ s.t. $\rightsquigarrow, \propto, \sigma \models H$ and $\gamma\sigma = t \xrightarrow{a} t'$ (resp. $\gamma\sigma = Pt$). Relations $\rightsquigarrow$ and $\propto$ are* supported *by G if, and only if, each of their (transition, resp. predicate) formulas are supported by a rule in $\mathcal{R}_G$.*

Lemma 3.2.8. *For each* preg *system G there exists a unique sound and supported transition relation, and a unique sound and supported predicate relation.*

Proof. We start by showing that $\forall t \in T(\Sigma_G)$, the following sets exist and are uniquely defined:

$$\rightsquigarrow(t) = \{(a, t') \mid t \xrightarrow{a} t',\ a \in \mathcal{A}\} \text{ and } \propto(t) = \{P \mid Pt,\ P \in \mathcal{P}\}$$

Let $t = f(t_1, \ldots, t_n) \in T(\Sigma_G)$. In order to determine $\rightsquigarrow(t)$ and $\propto(t)$, we exploit the properties:

1. $(a, t') \in \rightsquigarrow(t)$ if, and only if, $\exists R = \dfrac{H}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]} \in \mathcal{R}^{\mathcal{A}}$ and σ a substitution s.t.:
   (a) $\sigma(x_i) = t_i$ $(\forall i \in \{1, \ldots, n\})$
   (b) $C[\vec{x}, \vec{y}]\sigma = t'$
   (c) $\forall x_i \xrightarrow{a_{ij}} y_{ij} \in H$, it holds that $(a_{ij}, \sigma(y_{ij})) \in \rightsquigarrow(t_i)$
   (d) $\forall x_i \overset{b}{\nrightarrow} \in H$, it holds that $\nexists t_i'.(b, t_i') \in \rightsquigarrow(t_i)$
   (e) $\forall P_{ij} x_i \in H$, it holds that $P_{ij} \in \propto(t_i)$
   (f) $\forall \neg P_{ij} x_i \in H$, it holds that $P_{ij} \notin \propto(t_i)$
2. $P \in \propto(t)$ if, and only if, $\exists R = \dfrac{H}{P(f(\vec{x}))} \in \mathcal{R}^{\mathcal{P}}$ and σ a substitution s.t.:
   (a) $\sigma(x_i) = t_i$ $(\forall i \in \{1, \ldots, n\})$
   (b) $\forall x_i \xrightarrow{a_{ij}} y_{ij} \in H$, it holds that $(a_{ij}, \sigma(y_{ij})) \in \rightsquigarrow(t_i)$
   (c) $\forall x_i \overset{b}{\nrightarrow} \in H$, it holds that $\nexists t_i'.(b, t_i') \in \rightsquigarrow(t_i)$

(d) $\forall P_{ij}x_i \in H$, it holds that $P_{ij} \in \propto(t_i)$

(e) $\forall \neg P_{ij}x_i \in H$, it holds that $P_{ij} \notin \propto(t_i)$

We further determine $\rightsquigarrow(t)$ and $\propto(t)$ by induction on the structure of t.

- *Base case:* f is a constant symbol. For this case, conditions (1a) and (1c)–(1f) are all satisfied. Moreover, there are no occurrences of variables in the target, so the target has to be a closed term C. Therefore, we take $t' = C$ and consider (a, t') an element of the relation $\rightsquigarrow(t)$. Similarly for the predicate rules – $P \in \propto(t)$.

- *Induction step:* $t = f(t_1, \ldots, t_n)$. By the inductive hypothesis, the sets $\rightsquigarrow(t_i)$ and $\propto(t_i)$ exist $(\forall i \in \{1, \ldots, n\})$. So, identifying all the substitutions σ that satisfy the conditions in 1. and 2. would suffice to determine the elements in $\rightsquigarrow(t)$ and $\propto(t)$, respectively.

According to the reasoning above, it is obvious that $\rightsquigarrow(t)$ and $\propto(t)$ are unique. Now take $\rightarrow_G = \{(t, a, t') \mid t \in T(\Sigma_G),\ a \in \mathcal{A} \text{ and } (a, t') \in \rightsquigarrow(t)\}$ and $\ltimes_G = \cup_{t \in T(\Sigma_G)} \propto(t) \times \{t\}$. It follows immediately that, by construction, both $\rightarrow_G$ and $\ltimes_G$ are unique, sound and supported. □

Consider a *preg* system G. Formally, the operational semantics of the closed process terms in G is fully characterized by the relations $\rightarrow_G \subseteq T(\Sigma_G) \times \mathcal{A} \times T(\Sigma_G)$ and $\ltimes_G \subseteq \mathcal{P} \times T(\Sigma_G)$, called the (unique) *sound and supported* transition and, respectively, predicate relations. Intuitively, soundness guarantees that $\rightarrow_G$ and $\ltimes_G$ are closed with respect to the application of the rules in $\mathcal{R}_G$ on $T(\Sigma_G)$, *i.e.*, $\rightarrow_G$ (resp. $\ltimes_G$) contains the set of all possible transitions (resp. predicates) process terms in $T(\Sigma_G)$ can perform (resp. satisfy) according to $\mathcal{R}_G$. The requirement that $\rightarrow_G$ and $\ltimes_G$ be supported means that all the transitions performed (resp. all the predicates satisfied) by a certain process term can be "derived" from the deductive system described by $\mathcal{R}_G$. As a notational convention, we write $S \xrightarrow{a}_G S'$ and $P_G S$ whenever $(S, a, S') \in \rightarrow_G$ and $(P, S) \in \ltimes_G$. We omit the subscript G when it is clear from the context.

Lemma 3.2.9. *Let G be a* preg *system. Then, for each $t \in T(\Sigma_G)$ the set $\{(a, t') \mid t \xrightarrow{a} t',\ a \in \mathcal{A}\}$ is finite.*

Next we introduce the notion of *bisimilarity* – the equivalence over processes we consider in this chapter.

Definition 3.2.10 (Bisimulation)**.** *Consider a* preg *system $G = (\Sigma_G, \mathcal{R}_G)$. A symmetric relation $R \subseteq T(\Sigma_G) \times T(\Sigma_G)$ is a* bisimulation *if, and only if:*

1. *for all* $s, t, s' \in T(\Sigma_G)$*, whenever* $(s,t) \in R$ *and* $s \xrightarrow{a} s'$ *for some* $a \in \mathcal{A}$*, then there is some* $t' \in T(\Sigma_G)$ *such that* $t \xrightarrow{a} t'$ *and* $(s', t') \in R$*;*

2. *whenever* $(s,t) \in R$ *and* Ps $(P \in \mathcal{P})$ *then* Pt*.*

Two closed terms s *and* t *are* bisimilar *(written* $s \sim t$*) if, and only if, there is a bisimulation relation* R *such that* $(s,t) \in R$*.*

Proposition 3.2.11 ([30])**.** *Let* G *be a* preg *system. Then* $\sim$ *is an equivalence relation and a congruence for all operations* f *of* G*.*

Definition 3.2.12 (Disjoint extension)**.** *A* preg *system* G' *is a disjoint extension of a* preg *system* G*, written* $G \sqsubseteq G'$*, if the signature and the rules of* G' *include those of* G*, and* G' *does not introduce new rules for operations in* G*.*

It is well known that if $G \sqsubseteq G'$ then two terms in $T(\Sigma_G)$ are bisimilar in G if and only if they are bisimilar in G'.

From this point onward, our focus is to find a *sound and ground-complete axiomatization of bisimilarity on closed terms* for an arbitrary *preg* system G, *i.e.*, to identify a (finite) axiom system E_G so that $E_G \vdash s = t$ *iff* $s \sim t$ for all $s, t \in T(\Sigma_G)$. The method we apply is an adaptation of the technique in [6] to the *preg* setting. The strategy is to incrementally build a finite, head-normalizing axiomatization for general *preg* terms, *i.e.*, an axiomatization that, when applied recursively, reduces the completeness problem for arbitrary terms to that for synchronization trees. This way, the proof of ground-completeness for G reduces to showing the equality of closed tree terms.

3.3 Preliminary steps towards the axiomatization

In this section we start by identifying an appropriate language for expressing finite trees with predicates. We continue in the style of [6], by extending the language with a kind of restriction operator used for expressing the inability of a process to perform a certain action or to satisfy a given predicate. (This operator is used in the axiomatization of negative premises.) We provide the structural operational semantics of the resulting language, together with a sound and ground-complete axiomatization of bisimilarity on finite trees with predicates.

3.3.1 Finite trees with predicates

The language for trees we use in this chapter is an extension with predicates of the language BCCSP [79]. The syntax of BCCSP consists of closed terms built from a constant δ (*deadlock*), the binary operator $\_+\_$ (*nondeterministic choice*), and the unary operators $a.\_$ (*action prefix*), where a ranges over the actions in a set $\mathcal{A}$. Let $\mathcal{P}$ be a set of predicates. For each $P \in \mathcal{P}$ we consider a process constant κ_P, which "witnesses" the associated predicate in the definition of a process. Intuitively, κ_P stands for a process that only satisfies predicate P and has no transition.

A finite tree term t is built according to the following grammar:

$$t ::= \delta \mid \kappa_P \ (\forall P \in \mathcal{P}) \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t. \tag{3.1}$$

Intuitively, δ represents a process that does not exhibit any behaviour, $s + t$ is the nondeterministic choice between the behaviours of s and t, while $a.t$ is a process that first performs action a and behaves like t afterwards. The operational semantics that captures this intuition is given by the rules of BCCSP:

$$\frac{}{a.x \xrightarrow{a} x} \ (rl_1) \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \ (rl_2) \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \ (rl_3)$$

Figure 3.1: The semantics of BCCSP

As our goal is to extend BCCSP, the next step is to find an appropriate semantics for predicates. As can be seen in Figure 3.1, action performance is determined by the shape of the terms. Consequently, we choose to define predicates in a similar fashion.

Consider a predicate P and the term $t = \kappa_P$. As previously mentioned, the purpose of κ_P is to witness the satisfiability of P. Therefore, it is natural to consider that κ_P satisfies P.

Take for example the *immediate termination* predicate $\downarrow$. As a term $s + s'$ exhibits the behaviour of both s and s', it is reasonable to state that $(s + s') \downarrow$ if $s \downarrow$ or $s' \downarrow$. Note that for a term $t = a.t'$ the statement $t \downarrow$ is in contradiction with the meaning of immediate termination, since t can initially only execute action a. Predicates of this kind are called *explicit predicates* in what follows.

Consider now the *eventual termination* predicate $\lightning$. In this situation, it is proper to consider that $(s + t)\lightning$ if $s\lightning$ or $t\lightning$ and, moreover, that $a.s\lightning$ if $s\lightning$. We refer to predicates

such as $\lightning$ as *implicit predicates* (that range over a set $\mathcal{P}^I$ included in $\mathcal{P}$), since their satisfiability propagates through the structure of tree terms in an implicit fashion. We denote by $\mathcal{A}_P$ (included in $\mathcal{A}$) the set consisting of the actions a for which this behaviour is permitted when reasoning on the satisfiability of predicate P.

The rules expressing the semantics of predicates are:

$$\frac{}{P\kappa_P}\ (rl_4) \qquad \frac{Px}{P(x+y)}\ (rl_5) \qquad \frac{Py}{P(x+y)}\ (rl_6) \qquad \frac{Px}{P(a.x)}, \forall P \in \mathcal{P}^I\ \forall a \in \mathcal{A}_P\ (rl_7)$$

Figure 3.2: The semantics of predicates

The operational semantics of trees with predicates is given by the set of rules (rl_1)–(rl_7) illustrated in Figure 3.1 and Figure 3.2. In particular, δ satisfies no predicate.

For notational consistency, we make the following conventions. Let $\mathcal{A}$ be an action set and $\mathcal{P}$ a set of predicates. Σ_{FTP} represents the signature of finite trees with predicates. $T(\Sigma_{FTP})$ is the set of (closed) tree terms built over Σ_{FTP}, and $\mathcal{R}_{FTP}$ is the set of rules (rl_1)–(rl_7). Moreover, by FTP we denote the system $(\Sigma_{FTP}, \mathcal{R}_{FTP})$.

Discussion on the design decisions. At first sight, it seems reasonable for our framework to allow for language specifications containing rules of the shape $\frac{}{P(x+y)}$, or just one of (rl_5) and (rl_6). We decided, however, to disallow them, as their presence would invalidate standard algebraic properties such as the idempotence and the commutativity of $\_+\_$.

Without loss of generality we avoid rules of the form $\frac{}{P(a.x)}$. As far as the user is concerned, in order to express that $a.x$ satisfies a predicate P, one can always add the witness κ_P as a summand: $a.x + \kappa_P$. This decision helped us avoid some technical problems for the soundness and completeness proofs for the case of the restriction operator $\partial_{\mathcal{B},Q}$, which is presented in Section 3.3.3.

Due to the aforementioned restriction, we also had to leave out universal predicates with rules of the form $\frac{Px\ Py}{P(x+y)}$. However, the elimination of universal predicates is not a theoretical limitation to what one can express, since a universal predicate can always be defined as the negation of an existential one.

As a last approach, we thought of allowing the user to specify existential predicates using rules of the form $\frac{P_1x \ldots P_nx}{P(x+y)}(*)$ and $\frac{P_1y \ldots P_ny}{P(x+y)}(**)$ (instead of (rl_5) and (rl_6)). However, in order to maintain the validity of the axiom $x + x = x$ in the presence of rules

of these forms, it would have to be the case that one of the predicates P_i in the premises is P itself. (If that were not the case, then let t be the sum of the constants witnessing the P_i's for a rule of the form $(*)$ above with a minimal set of set premises. We have that $t + t$ satisfies P by rule $(*)$. On the other hand, Pt does not hold since none of the P_i is equal to P and no rule for P with a smaller set of premises exists.) Now, if a rule of the form $(*)$ has a premise of the form Px, then it is subsumed by (rl_5) which we must have to ensure the validity of laws such as $\kappa_P = \kappa_P + \kappa_P$.

3.3.2 Axiomatizing finite trees

In what follows we provide a finite sound and ground-complete axiomatization (E_{FTP}) for bisimilarity over finite trees with predicates.

The axiom system E_{FTP} consists of the following axioms:

$$x + y = y + x \quad (A_1) \qquad x + x = x \quad (A_3)$$
$$(x + y) + z = x + (y + z) \quad (A_2) \qquad x + \delta = x \quad (A_4)$$
$$a.(x + \kappa_P) = a.(x + \kappa_P) + \kappa_P, \forall P \in \mathcal{P}^{\mathcal{I}} \ \forall a \in \mathcal{A}_P \ (A_5)$$

Figure 3.3: The axiom system E_{FTP}

Axioms (A_1)–(A_4) are well-known [106]. Axiom (A_5) describes the propagation of witness constants for the case of implicit predicates.

We now introduce the notion of terms in *head normal form*. This concept plays a key role in the proofs of completeness for the axiom systems generated by our framework.

Definition 3.3.1 (Head Normal Form)**.** *Let Σ be a signature such that $\Sigma_{FTP} \subseteq \Sigma$. A term t in $T(\Sigma)$ is in* head normal form *(for short, h.n.f.) if*

$$t = \sum_{i \in I} a_i.t_i + \sum_{j \in J} \kappa_{P_j}, \textit{and the } P_j \textit{ are all the predicates satisfied by } t.$$

The empty sum ($I = \emptyset, J = \emptyset$) is denoted by the deadlock constant δ.

Lemma 3.3.2. *E_{FTP} is head normalizing for terms in $T(\Sigma_{FTP})$. That is, for all t in $T(\Sigma_{FTP})$, there exists t' in $T(\Sigma_{FTP})$ in h.n.f. such that $E_{FTP} \vdash t = t'$ holds.*

Proof. The reasoning is by induction on the structure of t. □

Theorem 3.3.3. *E_{FTP} is sound and ground-complete for bisimilarity on $T(\Sigma_{FTP})$. That is, it holds that $(\forall t, t' \in T(\Sigma_{FTP}))\,.\, E_{FTP} \vdash t = t'$ iff $t \sim t'$.*

The full proof of this theorem is included in Appendix 3.B.

3.3.3 Axiomatizing negative premises

A crucial step in finding a complete axiomatization for *preg* systems is the "axiomatization" of negative premises (of the shape $x \overset{a}{\nrightarrow}$, $\neg Px$). In the style of [6], we introduce the restriction operator $\partial_{\mathcal{B},\mathcal{Q}}$, where $\mathcal{B} \subseteq \mathcal{A}$ and $\mathcal{Q} \subseteq \mathcal{P}$ are the sets of initially forbidden actions and predicates, respectively. The semantics of $\partial_{\mathcal{B},\mathcal{Q}}$ is given by the two types of transition rules in Figure 3.4.

$$\frac{x \xrightarrow{a} x'}{\partial_{\mathcal{B},\mathcal{Q}}(x) \xrightarrow{a} \partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(x')} \text{ if } a \notin \mathcal{B}\ (rl_8) \qquad \frac{Px}{P(\partial_{\mathcal{B},\mathcal{Q}}(x))} \text{ if } P \notin \mathcal{Q}\ (rl_9)$$

Figure 3.4: The semantics of $\partial_{\mathcal{B},\mathcal{Q}}$

Note that $\partial_{\mathcal{B},\mathcal{Q}}$ behaves like the one step restriction operator in [6] for the actions in $\mathcal{B}$, as the restriction on the action set disappears after one transition. On the other hand, for the case of predicates in $\mathcal{Q}$, the operator $\partial_{\mathcal{B},\mathcal{Q}}$ resembles the CCS restriction operator [106] since, due to the presence of implicit predicates, not all the restrictions related to predicate satisfaction necessarily disappear after one step, as will become clear in what follows.

We write E^{∂}_{FTP} for the extension of E_{FTP} with the axioms involving $\partial_{\mathcal{B},\mathcal{Q}}$ presented in Figure 3.5. $\mathcal{R}^{\partial}_{FTP}$ stands for the set of rules (rl_1)–(rl_9), while FTP^{∂} represents the system $(\Sigma^{\partial}_{FTP}, \mathcal{R}^{\partial}_{FTP})$.

$$\partial_{\mathcal{B},\mathcal{Q}}(\delta) = \delta \quad (A_6) \qquad \partial_{\mathcal{B},\mathcal{Q}}(a.x) = \textstyle\sum_{P\notin\mathcal{Q}, P(a.x)} \kappa_P \text{ if } a \in \mathcal{B} \quad (A_9)$$

$$\partial_{\mathcal{B},\mathcal{Q}}(\kappa_P) = \delta \text{ if } P \in \mathcal{Q} \quad (A_7) \qquad \partial_{\mathcal{B},\mathcal{Q}}(a.x) = \partial_{\emptyset,\mathcal{Q}}(a.x) \text{ if } a \notin \mathcal{B} \quad (A_{10})$$

$$\partial_{\mathcal{B},\mathcal{Q}}(\kappa_P) = \kappa_P \text{ if } P \notin \mathcal{Q} \quad (A_8) \qquad \partial_{\emptyset,\mathcal{Q}}(a.x) = a.\partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(x) \quad (A_{11})$$

$$\partial_{\mathcal{B},\mathcal{Q}}(x + y) = \partial_{\mathcal{B},\mathcal{Q}}(x) + \partial_{\mathcal{B},\mathcal{Q}}(y)\ (A_{12})$$

Figure 3.5: The axiom system $E^{\partial}_{FTP} \setminus E_{FTP}$

Axiom (A_6) states that it is useless to impose restrictions on δ, as δ does not exhibit any behaviour. The intuition behind (A_7) is that since a predicate witness κ_P does

not perform any action, inhibiting the satisfiability of P leads to a process with no behaviour, namely δ. Consequently, if the restricted predicates do not include P, the resulting process is κ_P itself (see (A_8)). Inhibiting the only action a process $a.t$ can perform leads to a new process that, in the best case, satisfies some of the predicates in $\mathcal{P}^I$ satisfied by t (by (rl_7)) if $\mathcal{Q} \neq \mathcal{P}^I$ (see (A_9)). Whenever the restricted action set $\mathcal{B}$ does not contain the only action a process $a.t$ can perform, then it is safe to give up $\mathcal{B}$ (see (A_{10})). As a process $a.t$ only satisfies the predicates also satisfied by t, it is straightforward to see that $\partial_{\emptyset,\mathcal{Q}}(a.t)$ is equivalent to the process obtained by propagating the restrictions on implicit predicates deeper into the behaviour of t (see (A_{11})). Axiom (A_{12}) is given in conformity with the semantics of $\_+\_$ ($s + t$ encapsulates both the behaviours of s and t).

Remark 3.3.4. *For the sake of brevity and readability, in Figure 3.5 we presented* (A_9)*, which is a schema with infinitely many instances. However, it can be replaced by a finite family of axioms. See Appendix 3.C for details.*

Remark 3.3.5. *We can also combine* (A_{10}) *and* (A_{11}) *in order to obtain a single axiom,* $\partial_{\mathcal{B},\mathcal{Q}}(a.x) = a.\partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(x)$ *if* $a \notin \mathcal{B}$.

Theorem 3.3.6. *The following statements hold for* E^{∂}_{FTP}*:*

1. E^{∂}_{FTP} *is sound for bisimilarity on* $T(\Sigma^{\partial}_{FTP})$.
2. $\forall t \in T(\Sigma^{\partial}_{FTP}), \exists t' \in T(\Sigma_{FTP})$ *s.t.* $E^{\partial}_{FTP} \vdash t = t'$.

As proving completeness for FTP^{∂} can be reduced to showing completeness for FTP (already proved in Theorem 3.3.3), the following result is an immediate consequence of Theorem 3.3.6:

Corollary 3.3.7. E^{∂}_{FTP} *is sound and complete for bisimilarity on* $T(\Sigma^{\partial}_{FTP})$.

3.4 Smooth and distinctive operations

Recall that our goal is to provide a sound and ground-complete axiomatization for bisimilarity on systems specified in the *preg* format. As the *preg* format is too permissive for achieving this result directly, our next task is to find a class of operations for which we can build such an axiomatization by "easily" reducing it to the completeness result for FTP, presented in Theorem 3.3.3. In the literature, these operations are known as *smooth and distinctive* [6]. As we will see, these operations are incrementally identified by imposing suitable restrictions on *preg* rules. The standard procedure is to first find the *smooth* operations, based on which one determines the *distinctive* ones.

Definition 3.4.1 (Smooth operation)**.**

1. *A* preg *transition rule is* smooth *if it is of the following format:*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I^+\} \qquad \{P_i x_i \mid i \in J^+\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]} \qquad \frac{\{x_i \overset{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

   where

   (a) *I^+, J^+, I^-, J^- disjointly cover the set $L = \{1, \ldots, l\}$,*

   (b) *in the target $C[\vec{x}, \vec{y}]$ we allow only: y_i $(i \in I^+)$, x_i $(i \in I^- \cup J^-)$.*

2. *A* preg *predicate rule is* smooth *if it has the form above, its premises satisfy condition (1a) and its conclusion is $P(f(x_1, \ldots, x_l))$ for some $P \in \mathcal{P}$.*

3. *An operation f of a* preg *system is* smooth *if all its (transition and predicate) rules are smooth.*

By Definition 3.4.1, a rule ρ is smooth if it satisfies the following properties:

- a position i cannot be tested both positively and negatively at the same time,
- positions tested positively are either from I^+ or J^+ and they are not tested for the performance of multiple transitions (respectively, for the satisfiability of multiple predicates) within the same rule, and
- if ρ is a transition rule, then the occurrence of variables at positions $i \in I^+ \cup J^+$ is not allowed in the target of the consequent of ρ.

Remark 3.4.2. *Note that we can always consider a position i that does not occur as a premise in a rule for f as being negative, with the empty set of constraints (i.e. either $i \in I^-$ and $\mathcal{B}_i = \emptyset$, or $i \in J^-$ and $\mathcal{Q}_i = \emptyset$).*

Definition 3.4.3 (Distinctive operation)**.** *An operation f of a* preg *system is* distinctive *if it is smooth and:*

- *for each argument i, either all rules for f test i positively, or none of them does, and*
- *for any two distinct rules for f there exists a position i tested positively, such that one of the following holds:*
  - *both rules have actions that are different in the premise at position i,*
  - *both rules have predicates that are different in the premise at position i,*

- *one rule has an action premise at position i, and the other rule has a predicate test at the same position i.*

According to the first requirement in Definition 3.4.3, we state that for a smooth and distinctive operation f, a position i is *positive* (respectively, *negative*) for f if there is a rule for f such that i is tested positively (respectively, negatively) for that rule.

The existence of a family of smooth and distinctive operations "describing the behaviour" of a general *preg* operation is formalized by the following lemma:
Lemma 3.4.4. *Consider a* preg *system G. Then there exist a* preg *system G', which is a disjoint extension of G and FTP, and a finite axiom system E such that*

1. *E is sound for bisimilarity over any disjoint extension G'' of G', and*
2. *for each term t in $T(\Sigma_G)$ there is some term t' in $T(\Sigma_{G'})$ such that t' is built solely using smooth and distinctive operations and E proves $t = t'$.*

A detailed description of the transformation process from general *preg* to smooth and distinctive operations is provided in Appendix 3.E.

3.4.1 Axiomatizing smooth and distinctive *preg* operations

To start with, consider, for the good flow of the presentation, that we only handle explicit predicates (*i.e.*, we take $\mathcal{P}^I = \emptyset$). Towards the end of the section we discuss how to extend the presented theory to implicit predicates. We proceed in a similar fashion to [6] by defining a set of laws used in the construction of a complete axiomatization for bisimilarity on terms built over smooth and distinctive operations. The strength of these laws lies in their capability of reducing terms to their head normal form, thus reducing completeness for general *preg* systems to completeness of E_{FTP} (which has already been proved in Section 3.3.2).
Definition 3.4.5. *Let f be a smooth and distinctive l-ary operation of a* preg *system G, such that $FTP^\partial \sqsubseteq G$.*

1. *For a positive position $i \in L = \{1, \ldots, l\}$, the* distributivity law *for i w.r.t. f is given as follows:*

$$f(X_1, \ldots, X_i' + X_i'', \ldots, X_l) = f(X_1, \ldots, X_i', \ldots, X_l) + f(X_1, \ldots, X_i'', \ldots, X_l).$$

2. *For a rule $\rho \in \mathcal{R}$ for f the* trigger law *is, depending on whether ρ is a transition or a predicate rule:*

$$f(\vec{X}) = \begin{cases} c.C[\vec{X}, \vec{y}] & , \rho \in \mathcal{R}^{\mathcal{A}} \quad \text{(action law)} \\ \kappa_P & , \rho \in \mathcal{R}^{\mathcal{P}} \quad \text{(predicate law)} \end{cases}$$

where

$$X_i \equiv \begin{cases} a_i.y_i & , i \in I^+ \\ \kappa_{P_i} & , i \in J^+ \\ \partial_{\mathcal{B}_i,\mathcal{Q}_i}(x_i) & , i \in I^- \cup J^- \end{cases} .$$

3. *Suppose that for $i \in L$, term X_i is in one of the forms $\delta, z_i, \kappa_{P_i}, a.z_i, a.z_i + z_i'$ or $\kappa_{P_i} + z_i$. Suppose further that for each rule for f there exists $X_j \in \vec{X}$ ($j \in \{1, \ldots, l\}$) s.t. one of the following holds:*

   - *$j \in I^+$ and ($X_j \equiv \delta$ or $X_j \equiv b.z_j$ ($b \neq a_j$) or $X_j \equiv \kappa_Q$, for some Q),*
   - *$j \in J^+$ and ($X_j \equiv \delta$ or $X_j \equiv \kappa_Q$ ($Q \neq P_j$) or $X_j \equiv b.z_j$, for some b),*
   - *$j \in I^-$ and $X_j \equiv b.z_j + z_j'$, where $b \in \mathcal{B}_j$,*
   - *$j \in J^-$ and $X_j \equiv \kappa_Q + z_j$, where $Q \in \mathcal{Q}_j$.*

   Then the deadlock law *is as follows:*

$$f(\vec{X}) = \delta.$$

Example 3.4.6. *Consider the* right-biased sequential composition *operation $\_ ;^r \_$, whose semantics is given by the rules $\frac{x\downarrow \; y \xrightarrow{a} y'}{x ;^r y \xrightarrow{a} y'}$, $\frac{x\downarrow \; y\downarrow}{(x ;^r y)\downarrow}$, and $\frac{x\downarrow \; y\uparrow}{(x ;^r y)\uparrow}$, where $\downarrow$ and $\uparrow$ are, respectively, the* immediate termination *and* immediate divergence *predicates. $\_ ;^r \_$ is one of the auxiliary operations generated by the algorithm for deriving smooth and distinctive operations when axiomatizing the* sequential composition *in the presence of the two mentioned predicates.*

The laws derived according to Definition 3.4.5 for this system are:

$$\begin{array}{rclcrcl} (x+y) ;^r z & = & x ;^r z + y ;^r z & \qquad & \delta ;^r y & = & \delta \\ x ;^r (y+z) & = & x ;^r y + x ;^r z & & k_\uparrow ;^r y & = & \delta \\ k_\downarrow ;^r a.y & = & a.y & & a.x ;^r y & = & \delta \\ k_\downarrow ;^r k_\downarrow & = & k_\downarrow & & x ;^r \delta & = & \delta \\ k_\downarrow ;^r k_\uparrow & = & k_\uparrow & & \ldots & & \end{array}$$

Theorem 3.4.7. *Consider G a* preg *system such that $FTP^{\partial} \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G \setminus \Sigma^{\partial}_{FTP}$ be a collection of smooth and distinctive operations of G. Let E_G be the finite axiom system that extends E^{∂}_{FTP} with the following axioms for each $f \in \Sigma$:*

- *for each positive argument i of f, a distributivity law (Definition 3.4.5.1),*
- *for each transition rule for f, an action law (Definition 3.4.5.2),*
- *for each predicate rule for f, a predicate law (Definition 3.4.5.2), and*
- *all deadlock laws for f (Definition 3.4.5.3).*

The following statements hold for E_G, for any G' such that $G \sqsubseteq G'$:

1. *E_G is sound for bisimilarity on $T(\Sigma_{G'})$.*
2. *E_G is head normalizing for $T(\Sigma \cup \Sigma^{\partial}_{FTP})$.*

Obtaining the soundness of the action law (Definition 3.4.5.2) requires some care when allowing for specifications with implicit predicates ($\mathcal{P}^I \neq \emptyset$). Consider a scenario in which a transition rule for a smooth and distinctive operation f is of the form $\frac{H}{f(\vec{X}) \xrightarrow{c} C[\vec{X},\vec{y}]}$. Assume the closed instantiation $\vec{X} = \vec{s}$, $\vec{y} = \vec{t}$ and assume that $P(c.C[\vec{s},\vec{t}])$ holds for some predicate P in $\mathcal{P}^I$. This means that $P(C[\vec{s},\vec{t}])$ holds. In order to preserve the soundness of the action law, $P(f(\vec{s}))$ should also hold, but this is impossible since f is distinctive. One possible way of ensuring the soundness of the action law in the presence of implicit predicates is to stipulate some syntactic consistency requirements on the language specification. One sufficient requirement would be that if predicate rule $\frac{H'}{P(C[\vec{z},\vec{y}])}$ is derivable, then the system should contain a predicate rule $\frac{H''}{P(f[\vec{z}])}$ with $H'' \subseteq H'$. This is enough to guarantee that if the right-hand side of the action law satisfies P then so does the left-hand side.

3.5 Soundness and completeness

Let us summarize our results so far. By Theorem 3.4.7, it follows that, for any *preg* system $G \sqsupseteq FTP^{\partial}$, there is an axiomatization that is head normalizing for $T(\Sigma \cup \Sigma^{\partial}_{FTP})$, where $\Sigma \subseteq \Sigma_G \setminus \Sigma^{\partial}_{FTP}$ is a collection of smooth and distinctive operations of G. Also, as hinted in Section 3.4 (Lemma 3.4.4), there exists a sound algorithm for transforming general *preg* operations to smooth and distinctive ones.

So, for any *preg* system G, we can build a *preg* system $G' \sqsupseteq G$ and an axiomatization $E_{G'}$ that is head normalizing for $T(\Sigma_{G'})$. This statement is formalized as follows:

Theorem 3.5.1. *Let G be a* preg *system. Then there exist $G' \sqsupseteq G$ and a finite axiom system $E_{G'}$ such that*

1. *$E_{G'}$ is sound for bisimilarity on $T(\Sigma_{G'})$,*
2. *$E_{G'}$ is head normalizing for $T(\Sigma_{G'})$,*

and moreover, G' and $E_{G'}$ can be effectively constructed from G.

Proof. The result follows immediately by Theorem 3.4.7 and by the existence of an algorithm used for transforming general *preg* to smooth and distinctive operations. □

Remark 3.5.2. *Theorem 3.5.1 guarantees ground-completeness of the generated axiomatization for well-founded* preg *specifications, that is,* preg *specifications in which each process can only exhibit finite behaviour.*

Let us further recall an example given in [6]. Consider the constant ω, specified by the rule $\omega \xrightarrow{a} \omega$. Obviously, the corresponding action law $\omega = a.\omega$ will apply for an infinite number of times in the normalization process. So the last step in obtaining a complete axiomatization is to handle infinite behaviour.

Let t and t' be two processes with infinite behaviour (remark that the infinite behaviour is a consequence of performing actions for an infinite number of times, so the extension to predicates is not a cause for this issue). Since we are dealing with finitely branching processes, it is well known that if two process terms are bisimilar at each finite depth, then they are bisimilar. One way of formalizing this requirement is to use the well-known *Approximation Induction Principle* (AIP) [28, 41].

Let us first consider the operations $\pi_n(\cdot)$, $n \in \mathbb{N}$, known as *projection operations*. The purpose of these operations is to stop the evolution of processes after a certain number of steps. The AIP is given by the following conditional equation:

$$x = y \text{ if } \pi_n(x) = \pi_n(y)\ (\forall n \in \mathbb{N}).$$

We further adapt the idea in [6] to our context, and model the infinite family of projection operations $\pi_n(\cdot)$, $n \in \mathbb{N}$, by a binary operation $\cdot/\cdot$ defined as follows:

$$\frac{x \xrightarrow{a} x' \quad h \xrightarrow{c} h'}{x/h \xrightarrow{a} x'/h'}\ (rl_{10}) \qquad \frac{Px}{P(x/h)}\ (rl_{11})$$

where c is an arbitrary action. Note that $\cdot/\cdot$ is a smooth and distinctive operation.

The role of variable h is to "control" the evolution of a process, *i.e.*, to stop the process in performing actions, after a given number of steps. Variable h (the "hourglass" in [6]) will always be instantiated with terms of the shape c^n, inductively defined as: $c^0 = \delta$, $c^{n+1} = c.c^n$.

Let $G = (\Sigma_G, \mathcal{R}_G)$ be a *preg* system. We use the notation $G_/$ to refer to the *preg* system $(\Sigma_G \cup \{\cdot/\cdot\}, \mathcal{R}_G \cup \{(rl_{10}), (rl_{11})\})$ – the extension of G with $\cdot/\cdot$. Moreover, we use the notation E_{AIP} to refer to the axioms for the smooth and distinctive operation $\cdot/\cdot$, derived as in Section 3.4.1 – Definition 3.4.5.

Based on the fact that $\cdot/\cdot$ is a smooth and distinctive operation, we derive the following axioms, as in Section 3.4.1 – Definition 3.4.5:

$$(x + y)/z = x/z + y/z \quad (A_{13})$$
$$x/(y + z) = x/y + x/z \quad (A_{14})$$
$$a.x/c.y = a.(x/y) \quad (A_{15})$$
$$\delta/y = \delta \quad (A_{16})$$
$$a.x/\delta = \delta \quad (A_{17})$$
$$\kappa_P/y = \kappa_P \quad (A_{18})$$

Figure 3.6: The axiom system E_{AIP}

We reformulate AIP according to the new operation $\cdot/\cdot$:

$$x = y \text{ if } x/c^n = y/c^n\ (\forall n \in \mathbb{N})$$

Lemma 3.5.3. *AIP is sound for bisimilarity on* $T(\Sigma_{FTP_/})$.

The whole proof can be found in Section 3.H.

Remark 3.5.4. *Note that axiom* $x/\delta = \delta$ *(51) in [6] is not sound for our approach, as by* (rl_{11}) x/δ *satisfies all the predicates satisfied by* x*, while by definition,* δ *satisfies no predicates. For the case of* preg *systems, axiom (51) is "encoded" by* (A_{17})*, and* (A_{18}) *for* $y = \delta$.

In what follows we provide the final ingredients for proving the existence of a ground-complete axiomatization for bisimilarity on *preg* systems. As previously stated, this is achieved by reducing completeness to proving equality in *FTP*. So, based on AIP, it would suffice to show that for any closed process term t and natural number n, there exists an *FTP* term equivalent to t at moment n in time:

Lemma 3.5.5. *Consider G a* preg *system. Then there exist $G' \sqsupseteq G_{/}$ and $E_{G'}$ with the property: $\forall t \in T(\Sigma_{G'}), \forall n \in \mathbb{N}, \exists t' \in T(\Sigma_{FTP})$ s.t. $E_{G'} \vdash t/c^n = t'$.*

At this point we can prove the existence of a sound and ground-complete axiomatization for bisimilarity on general *preg* systems:

Theorem 3.5.6 (Soundness and Completeness)**.** *Consider G a* preg *system. Then there exist $G' \sqsupseteq G_{/}$ and $E_{G'}$ a finite axiom system, such that $E_{G'} \cup E_{AIP}$ is sound and complete for bisimilarity on $T(\Sigma_{G'})$.*

3.6 Motivation for handling predicates as first-class notions

In the literature on the theory of rule formats for Structural Operational Semantics (especially, the work devoted to congruence formats for various notions of bisimilarity), predicates are often neglected at first and are only added to the considerations at a later stage. The reason is that one can encode predicates quite easily by means of transition relations. One can find a number of such encodings in the literature—see, for instance, [61, 138]. In each of these encodings, a predicate P is represented as a transition relation $\xrightarrow{P}$ (assuming that P is a fresh action label) with a fixed constant symbol as target. Using this translation, one can axiomatize bisimilarity over *preg* language specifications by first converting them into "equivalent" standard GSOS systems, and then applying the algorithm from [6] to obtain a finite axiomatization of bisimilarity over the resulting GSOS system.

In light of this approach, it is natural to wonder whether it is worthwhile to develop an algorithm to axiomatize *preg* language specifications directly. One possible answer, which has been presented several times in the literature [138], is that often one does not want to encode a language specification with predicates using one with transitions only. Sometimes, specifications using predicates are the most natural ones to write, and one should not force a language designer to code pred-

icates using transitions. (However, one can write a tool to perform the translation of predicates into transitions, which can therefore be carried out transparently to the user/language designer.) Also, developing an algorithm to axiomatize GSOS language specifications with predicates directly yields insight into the difficulties that result from the first-class use of, and the interplay among, various types of predicates, as far as axiomatizability problems are concerned. These issues would be hidden by encoding predicates as transitions. Moreover, the algorithm resulting from the encoding would generate axioms involving predicate-prefixing operators, which are somewhat unintuitive.

Naturalness is, however, often in the eye of the beholder. Therefore, we now provide a more technical reason why it may be worthwhile to develop techniques that apply to GSOS language specifications with predicates as first-class notions, such as the *preg* ones. Indeed, we now show how, using predicates, one can convert any standard GSOS language specification G into an equivalent *positive* one with predicates G^+.

Given a GSOS language G, the system G^+ will have the same signature and the same set of actions as G, but uses predicates cannot(a) for each action a. The idea is simply that "x cannot(a)" is the predicate formula that expresses that "x does not afford an a-labelled transition". The translation works as follows.

1. Each rule in G is also a rule in G^+, but one replaces each negative premise in each rule with its corresponding positive predicate premise. This means that $x \overset{a}{\nrightarrow}$ becomes x cannot(a).
2. One adds to G^+ rules defining the predicates cannot(a), for each action a. This is done in such a way that p cannot(a) holds in G^+ exactly when $p \overset{a}{\nrightarrow}$ in G, for each closed term p and action a. More precisely, we proceed as follows.
   (a) For each constant symbol f and action a, add the rule

   $$\frac{}{f \text{ cannot}(a)}$$

   whenever there is no transition rule in G with f as principal operation and with an a-labelled transition as its consequent.
   (b) For each operation f with arity at least one and action a, let $R(f, a)$ be the set of rules in G that have f as principal operation and an a-labelled transition as consequent. We want to add rules for the predicate cannot(a)

to G^+ that allow us to prove the predicate formula $f(p_1,\ldots,p_l)\,\mathrm{cannot}(a)$ exactly when $f(p_1,\ldots,p_l)$ does not afford an a-labelled transition in G. This occurs if, for each rule in $R(f,a)$, there is some premise that is not satisfied when the arguments of f are $p_1,\ldots,p_l$. To formalize this idea, let $H(R(f,a))$ be the collection of premises of rules in $R(f,a)$. We say that a choice function is a function $\phi : R(f,a) \to H(R(f,a))$ that maps each rule in $R(f,a)$ to one of its premises. Let

$$\begin{aligned} \mathrm{neg}(x \xrightarrow{a} x') &= x\,\mathrm{cannot}(a) \quad \text{and} \\ \mathrm{neg}(x \overset{a}{\nrightarrow}) &= x \xrightarrow{a} x', \quad \text{for some } x'. \end{aligned}$$

Then, for each choice function ϕ, we add to G^+ a predicate rule of the form

$$\frac{\{\mathrm{neg}(\phi(\xi)) \mid \xi \in R(f,a)\}}{f(x_1,\ldots,x_l)\,\mathrm{cannot}(a)},$$

where the targets of the positive transition formulae in the premises are chosen to be all different.

The above construction ensures the validity of the following lemma.

Lemma 3.6.1. *For each closed term p and action a,*

1. *$p \xrightarrow{a} p'$ in G if, and only if, $p \xrightarrow{a} p'$ in G^+;*
2. *p cannot(a) in G^+ if, and only if, $p \overset{a}{\nrightarrow}$ in G^+ (and therefore in G).*

This means that two closed terms are bisimilar in G if, and only if, they are bisimilar in G^+. Moreover, two closed terms are bisimilar in G^+ iff they are bisimilar when we only consider the transitions (and not the predicates cannot(a)).

The language G^+ modulo bisimilarity can be axiomatized using our algorithm without the need for the exponentially many restriction operators. The conversion to positive GSOS with predicates discussed above does incur in an exponential blow-up in the number of rules, but it gives an alternative way of generating ground-complete axiomatizations for standard GSOS languages to the one proposed in [6]. In general, it is useful to have several approaches in one's toolbox, since one may choose the one that is "less expensive" for the specific task at hand. Moreover, using positive GSOS operations, one can also try to extend the methods from the full version of the paper [2] (see Section 7.1 in the technical report available at `http://www.ru.is/~luca/PAPERS/cs011994.ps`) to optimize these axiomatizations.

It is worth noting that the predicates cannot(a) are not implicit, therefore the restrictions presented at the end of Section 3.4.1 need not to be imposed.

3.7 Conclusions and future work

In this chapter we have introduced the *preg* rule format, a natural extension of GSOS with arbitrary predicates. Moreover, we have provided a procedure (similar to the one in [6]) for deriving sound and ground-complete axiomatizations for bisimilarity of systems that match this format. In the current approach, explicit predicates are handled by considering constants witnessing their satisfiability as summands in tree expressions. Consequently, there is no explicit predicate P satisfied by a term of shape $\Sigma_{i\in I} a_i.t_i$.

The procedure introduced in this chapter has also enabled the implementation of a tool, presented in Chapter 7, that can be used to automatically reason on bisimilarity of systems specified as terms built over operations defined by *preg* rules.

Several possible extensions are left as future work. It would be worth investigating the properties of positive *preg* languages. By allowing only positive premises we eliminate the need of the restriction operators ($\partial_{\mathcal{B},Q}$) during the axiomatization process. This would enable us to deal with more general predicates over trees, such as those that may be satisfied by terms of the form $a.t$ where a ranges over some subset of the collection of actions.

Another direction for future research is that of understanding the presented work from a coalgebraic perspective. The extensions from [6] to the present chapter, might be thought as an extension from coalgebras for a functor $\mathscr{P}(\mathcal{A} \times Id)$ to a functor $\mathscr{P}(\mathcal{P}) \times \mathscr{P}(\mathcal{A} \times Id)$ where $\mathscr{P}$ is the powerset functor, $\mathcal{A}$ is the set of actions and $\mathcal{P}$ is the set of predicates. Also the language *FTP* coincides, apart from the recursion operator, with the one that would be obtained for the functor $\mathscr{P}(\mathcal{P}) \times \mathscr{P}(\mathcal{A} \times Id)$ in the context of Kripke polynomial coalgebras [51].

Finally, we plan to extend our axiomatization theory in order to reason on the bisimilarity of guarded recursively defined terms, following the line presented in [2].

3.A Proof of Lemma 3.3.2

Proof. The reasoning is by induction on the structure of t.

Base cases
– $t = \delta \Longrightarrow t' = \delta$ (h.n.f) $\Longrightarrow E_{FTP} \vdash t = t'$ (by reflexivity)
– $t = \kappa_P \Longrightarrow t' = \kappa_P$ (h.n.f.) $\Longrightarrow E_{FTP} \vdash t = t'$ (by reflexivity)

Inductive step cases
– $t = a.t'$; t' has a simpler structure than t, so, by the inductive hypothesis, $\exists t'_1$ in h.n.f. s.t. $E_{FTP} \vdash t' = t'_1$. As $\vdash$ is a congruence, $E_{FTP} \vdash a.t' = a.t'_1$, so $E_{FTP} \vdash t = a.t'_1$. Now, $P(t)$ holds for some predicate P if, and only if, $P(t')$ and $P \in \mathcal{P}^I$. Since t'_1 is the h.n.f. for t', it has the summand κ_P for each P s.t. $P(t')$. By using the instance of (A_5) for those predicates P satisfied by t' for which there is a rule of form (rl_7) for P, we add a κ_P summand to the h.n.f. for t. We obtain, therefore, $E_{FTP} \vdash t = a.t'_1 + \sum_{j \in J} \kappa_{P_j}$ which is in h.n.f., where $\{P_j \mid j \in J\}$ is the set of predicates in $\mathcal{P}^I$ that are satisfied by t'.
– $t = t_1 + t_2$, so $\exists t'_1, t'_2$ in h.n.f. s.t. $E_{FTP} \vdash t_i = t'_i, i \in \{1, 2\}$. By the congruence of $\vdash$ we have $E_{FTP} \vdash t_1 + t_2 = t'_1 + t'_2$. Using $(A_1), (A_2)$, we infer $E_{FTP} \vdash t_1 + t_2 = t'_{12}$, which is in h.n.f.

□

3.B Proof of Theorem 3.3.3

We first introduce a lemma that will be used in our proof:
Lemma 3.B.1. *Let G be a* preg *system. If*

- $t_1, t_2 \in T(\Sigma_G)$ *s.t.* $t_1 \xrightarrow{a} t$ *iff* $t_2 \xrightarrow{a} t$ *for all* $a \in \mathcal{A}$ *and for all* $t \in T(\Sigma_G)$
- Pt_1 *iff* Pt_2 *for any* $P \in \mathcal{P}$

then $t_1 \sim t_2$.

Proof. Follows directly from Definition 3.2.10 and the reflexivity of $\sim$. □

We further continue with the proof of Theorem 3.3.3.

Proof.

The soundness ($E_{FTP} \vdash s = t \Longrightarrow s \sim t$) follows in a standard fashion.

Proving $s \sim t \Longrightarrow E_{FTP} \vdash s = t$ [completeness]

Let us first define the function *height* that computes the height of the syntactic tree associated to a term $t \in T(\Sigma_{FTP})$:

$$height(t) = \begin{cases} 0 & \text{if } t \in \{\delta\} \cup \{\kappa_P \mid P \in \mathcal{P}\} \\ 1 + \max(height(t_1), height(t_2)) & \text{if } t = t_1 + t_2 \\ 1 + height(t') & \text{if } t = a.t' \end{cases}$$

Note that we may assume (by Lemma 3.3.2) that s, t are in head normal form.
We prove the property by induction on $M = \max(height(s), height(t))$.

Base case

- $M = 0 \Longrightarrow \begin{cases} \text{either} & s = t = \delta, \text{ so } E_{FTP} \vdash s = t \\ \text{or} & s = t = \kappa_P, \text{ so } E_{FTP} \vdash s = t \end{cases}$

Inductive step case($M > 0$) We show that $t = s + t$. In order to do that we argue that each summand of s is provably equal to a summand of t.

- let $a.s'$ be a summand of $s \Longrightarrow s \xrightarrow{a} s'$. As $s \sim t$, it follows that $t \xrightarrow{a} t'$ for some t' and $s' \sim t'$. Now $\max(height(s'), height(t')) < M \overset{i.h.}{\Longrightarrow} E_{FTP} \vdash s' = t'$, hence $E_{FTP} \vdash a.s' = a.t'$;
- let κ_P be a summand of s (Ps holds). As $s \sim t$, Pt. t is in h.n.f., therefore κ_P is also a summand of t.

So, every summand of s can be proved equal to a summand of t, so $E_{FTP} \vdash t = s + t$. (*)

By symmetry it follows that $s = t + s$ (**).

By (*,**), the congruence of $\vdash$ and $(A_1 - A_3)$, we have $E_{FTP} \vdash s = t$. □

Notice that we do not explicitly use (A_5) when proving the completeness of the axiom system E_{FTP}. This is because we work with processes in h.n.f.

3.C Axiom (A_9), a schema with infinitely many instances

In what follows we provide a finite family of axioms that can replace (A_9):

$$\partial_{\mathcal{B},\mathcal{Q}}(a.\delta) = \delta \text{ if } a \in \mathcal{B} \tag{$A_{9.1}$}$$

$$\partial_{\mathcal{B},\mathcal{Q}}(a.\kappa_P) = \begin{cases} \kappa_P & \text{if } a \in \mathcal{B} \text{ and } (P \in \mathcal{P}^{\mathcal{I}} \text{ and } P \notin \mathcal{Q} \text{ and } a \in \mathcal{A}_P) \\ \delta & \text{if } a \in \mathcal{B} \text{ and } (P \notin \mathcal{P}^{\mathcal{I}} \text{ or } P \in \mathcal{Q} \text{ or } a \notin \mathcal{A}_P) \end{cases} \tag{$A_{9.2}$}$$

$$\partial_{\mathcal{B},\mathcal{Q}}(a.b.x) = \partial_{\mathcal{A},\mathcal{Q}}(x) \text{ if } a \in \mathcal{B} \tag{$A_{9.3}$}$$

$$\partial_{\mathcal{B},\mathcal{Q}}(a.(x + y)) = \partial_{\mathcal{B},\mathcal{Q}}(a.x) + \partial_{\mathcal{B},\mathcal{Q}}(a.y) \text{ if } a \in \mathcal{B} \tag{$A_{9.4}$}$$

Figure 3.7: Axioms replacing (A_9)

Note that replacing (A_9) with $(A_{9.1})$ – $(A_{9.4})$ does not affect the validity of the statements in Theorem 3.3.6.

Soundness (statement 1) follows in a standard fashion, similar to Theorem 3.3.3.

In order to prove statement 2 we proceed as follows. Assume t is a tree term and $a \in \mathcal{B}$. We prove that there is a h.n.f t' s.t. $E^{\partial}_{FTP} \vdash \partial_{\mathcal{B},\mathcal{Q}}(a.t) = t'$, using $(A_{9.1})$ – $(A_{9.4})$. The result follows by induction on the structure of t:

- $t = \delta$. Then use $(A_{9.1})$.
- $t = \kappa_P$. Then use $(A_{9.2})$.
- $t = b.t''$. Use $(A_{9.3})$ and induction.
- $t = t_1 + t_2$. Use $(A_{9.4})$ and induction.

3.D Proof of Theorem 3.3.6

Proof.

Proving $E^{\partial}_{FTP} \vdash s = t \Longrightarrow s \sim t$ [soundness]

The property follows directly for $(A_{6,7,8})$ because neither the lhs, nor the rhs terms can "advance" with actions and their (in)satisfiability of predicates is easily check-

able.

Take (A_9). Let $s = \partial_{\mathcal{B},\mathcal{Q}}(a.\bar{s})$, with $a \in \mathcal{B}$. Prove that $s \sim t = \sum_{P \notin \mathcal{Q}, P(a.\bar{s})} \kappa_P$.

- As $a \in \mathcal{B}$ it follows that $s \overset{a}{\nrightarrow}$. Also $s \overset{b}{\nrightarrow}$ for any $b \in Act, b \neq a$. As t is a sum of constants, $t \overset{b}{\nrightarrow}$ for any $t \in Act$.
- If $Ps \overset{rl_9}{\Longrightarrow} P \notin \mathcal{Q}$ and $P(a.\bar{s})$. Then Pt also holds because κ_P is one of its summands. For the other direction, if Pt, then $P(a.\bar{s})$ and $P \notin \mathcal{Q} \overset{rl_9}{\Longrightarrow} P(\partial_{\mathcal{B},\mathcal{Q}}(a.\bar{s}))$.

Therefore, by Lemma 3.B.1, it follows that $s \sim t$.

Take (A_{10}). Let $s = \partial_{\mathcal{B},\mathcal{Q}}(a.\bar{s})$ and $a \notin \mathcal{B}$. Prove that $s \sim t = \partial_{\emptyset,\mathcal{Q}}(a.\bar{s})$.

- If $s \overset{b}{\rightarrow} s'$ for some fixed $b \in Act$ and $s' \in S$, then by (rl_8), it follows that $b \notin \mathcal{B}$, $s' = \partial_{\emptyset,\mathcal{Q}}(s'')$, and $a.\bar{s} \overset{b}{\rightarrow} s''$. It follows $b = a$, $s'' = \bar{s}$, and, therefore $s = \partial_{\mathcal{B},\mathcal{Q}}(a.\bar{s}) \overset{a}{\rightarrow} \partial_{\emptyset,\mathcal{Q}}(\bar{s})$. Now, does $t = \partial_{\emptyset,\mathcal{Q}}(a.\bar{s}) \overset{a}{\rightarrow} \partial_{\emptyset,\mathcal{Q}}(\bar{s})$ hold ? Yes, it does, because $a \notin \emptyset$ and (rl_8) can be used. Similarly for $t \overset{b}{\rightarrow} t'$.
- If $P(s = \partial_{\mathcal{B},\mathcal{Q}}(a.\bar{s}))$, then $\overset{(rl_9)}{\Longrightarrow} P \notin \mathcal{Q}$ and $P(a.\bar{s}) \overset{(rl_9)}{\Longrightarrow} P(\partial_{\emptyset,\mathcal{Q}}(a.\bar{s}) = t)$. Similarly for Pt.

Therefore, by Lemma 3.B.1, it follows that $s \sim t$.

Take (A_{11}). Let $s = \partial_{\emptyset,\mathcal{Q}}(a.\bar{s})$. Prove that $s \sim t = a.\partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(\bar{s})$.

- If $s \overset{b}{\rightarrow} s'$ for some fixed $b \in Act$ and $s' \in S$, then by (rl_8), it follows that $b \notin \mathcal{B}$, $s' = \partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(s'')$, and $a.\bar{s} \overset{b}{\rightarrow} s''$. It follows that $b = a$, $s'' = \bar{s}$, and, therefore $s = \partial_{\emptyset,\mathcal{Q}}(a.\bar{s}) \overset{a}{\rightarrow} \partial_{\emptyset,\mathcal{Q}\cup\mathcal{P}^I}(\bar{s})$. Now, by (rl_1) it holds that $t = a.\partial_{\mathcal{Q}\cap\mathcal{P}^I}(\bar{s}) \overset{a}{\rightarrow} \partial_{\mathcal{Q}\cap\mathcal{P}^I}(\bar{s})$. Similarly for $t \overset{b}{\rightarrow} t'$.
- If $P(s = \partial_{\emptyset,\mathcal{Q}}(a.\bar{s}))$, then $\overset{(rl_9)}{\Longrightarrow} P \notin \mathcal{Q}$ and $P(a.\bar{s}) \overset{(rl_7)}{\Longrightarrow} P \in \mathcal{P}^I$ and $P\bar{s} \overset{(rl_9)}{\Longrightarrow} P \in \mathcal{P}^I$ and $P(\partial_{\mathcal{Q}\cap\mathcal{P}^I}(\bar{s})) \overset{(rl_7)}{\Longrightarrow} P(a.\partial_{\mathcal{Q}\cap\mathcal{P}^I}(\bar{s}) = t)$. Similarly for Pt.

Therefore, by Lemma 3.B.1, it follows that $s \sim t$.

Take (A_{12}). Let $s = \partial_{\mathcal{B},\mathcal{Q}}(s_1 + s_2)$. Prove that $s \sim t = \partial_{\mathcal{B},\mathcal{Q}}(s_1) + \partial_{\mathcal{B},\mathcal{Q}}(s_2)$.

- If $s \overset{a}{\rightarrow} s'$ for some fixed $a \in Act$ and $s' \in S$, then by (rl_8), it follows that $a \notin \mathcal{B}$, $s' = \partial_{\emptyset,\mathcal{Q}}(s'')$, and $s_1 + s_2 \overset{a}{\rightarrow} s''$. Then $s_1 \overset{a}{\rightarrow} s''$ or $s_2 \overset{a}{\rightarrow} s''$ (by $(rl_{2,3})$),

so $\partial_{\mathcal{B},\mathcal{Q}}(s_1) \xrightarrow{a} \partial_{\emptyset,\mathcal{Q}}(s'')$ or $\partial_{\mathcal{B},\mathcal{Q}}(s_2) \xrightarrow{a} \partial_{\emptyset,\mathcal{Q}}(s'')$ (by (rl_8)). Finally $\overset{(rl_{2,3})}{\Longrightarrow} \partial_{\mathcal{B},\mathcal{Q}}(s_1) + \partial_{\mathcal{B},\mathcal{Q}}(s_2) \xrightarrow{a} s'$. Similarly for $t \xrightarrow{a} t'$.

- Also, if Ps for some fixed $P \in \mathcal{P}$, then by (rl_9) it follows that $P \notin \mathcal{Q}$ and $P(s_1 + s_2)$. Then by $(rl_{2,3})$ and (rl_9) it follows that $P(\partial_{\mathcal{B},\mathcal{Q}}(s_1))$ or $P(\partial_{\mathcal{B},\mathcal{Q}}(s_2))$. Finally $\overset{(rl_{2,3})}{\Longrightarrow} P(\partial_{\mathcal{B},\mathcal{Q}}(s_1 + s_2))$. Similarly for Pt.

Therefore, by Lemma 3.B.1, it follows that $s \sim t$.

We have covered all the possibilities, so E^{∂}_{FTP} is sound for bisimilarity on $T(\Sigma^{\partial}_{FTP})$.

Proving $\forall t \in T(\Sigma^{\partial}_{FTP}), \exists t' \in T(\Sigma_{FTP})$ *s.t.* $E^{\partial}_{FTP} \vdash t = t'$

Note that in order to prove the statement above, it suffices to show that E^{∂}_{FTP} can be used to eliminate all the occurrences of ∂ from $\partial_{\mathcal{B},\mathcal{Q}}(t)$, for all closed terms $t \in T(\Sigma_{FTP})$, where $\mathcal{B}, \mathcal{Q}$ are two sets of restricted actions and, respectively, predicates. We proceed by structural induction on the structure of t.

Base cases

- $t = \delta \overset{(A_6)}{\Longrightarrow} E^{\partial}_{FTP} \vdash \partial_{\mathcal{B},\mathcal{Q}}(\delta) = \delta$;
- $t = \kappa_P \overset{(A_{7,8})}{\Longrightarrow} E^{\partial}_{FTP} \vdash \partial_{\mathcal{B},\mathcal{Q}}(\kappa_P) = \begin{cases} \delta & \text{if } P \in \mathcal{Q} \\ \kappa_P & \text{if } P \notin \mathcal{Q} \end{cases}$.

Inductive step cases

- $t = t_1 + t_2 \overset{(A_{12})}{\Longrightarrow} \partial_{\mathcal{B},\mathcal{Q}}(t_1 + t_2) = \partial_{\mathcal{B},\mathcal{Q}}(t_1) + \partial_{\mathcal{B},\mathcal{Q}}(t_2)$. Both t_1 and t_2 are subterms of t, so, by the inductive hypothesis, $\exists t'_1, t'_2 \in T(\Sigma_{FTP})$ such that $E^{\partial}_{FTP} \vdash \begin{cases} \partial_{\mathcal{B},\mathcal{Q}}(t_1) = t'_1 \\ \partial_{\mathcal{B},\mathcal{Q}}(t_2) = t'_2 \end{cases} \overset{\text{congr.} \vdash}{\Longrightarrow} E^{\partial}_{FTP} \vdash \partial_{\mathcal{B},\mathcal{Q}}(t_1) + \partial_{\mathcal{B},\mathcal{Q}}(t_2) = t'_1 + t'_2 \in T(\Sigma_{FTP})$;
- $t = a.t'$
  - $a \in \mathcal{B} \overset{(A_9)}{\Longrightarrow} \partial_{\mathcal{B},\mathcal{Q}}(a.t') = \sum_{P \notin \mathcal{Q}, P(a.x)} \kappa_P \in T(\Sigma_{FTP})$;
  - $a \notin \mathcal{B} \overset{(A_{10})}{\Longrightarrow} \partial_{\mathcal{B},\mathcal{Q}}(a.t') = \partial_{\emptyset,\mathcal{Q}}(a.t') \overset{(A_{11})}{=} a.\partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^I}(t')$. As t' is a subterm of t, by the inductive hypothesis it follows that $\exists t'' \in T(\Sigma_{FTP})$ s.t. $E^{\partial}_{FTP} \vdash a.\partial_{\mathcal{Q}\cap\mathcal{P}^I}(t') = a.t'' \in T(\Sigma_{FTP})$.

We have covered all the possibilities, so E^{∂}_{FTP} eliminates all the occurrences of ∂ from terms in $T(\Sigma^{\partial}_{FTP})$. □

3.E From general *preg* to smooth and distinctive

Given an arbitrary *preg* l-ary operation f, producing a family of smooth and distinctive operations capturing the behaviour of f consists of the following steps.

Transforming general *preg* to smooth operations. According to Definition 3.4.1, an operation f is not smooth if there exists (at least) one rule ρ for f that matches (at least) one of the following conditions:

- there are clashes between the sets I^+, I^-, J^+ and J^-,
- there is a positive position that has multiple transitions or that satisfies multiple predicates or
- ρ is a transition rule that contains in its target variables on positive positions.

By way of example, a rule satisfying all the items above is:

$$\rho = \frac{x \xrightarrow{a} y_1 \quad x \xrightarrow{b} y_2 \quad x \overset{d}{\nrightarrow} \quad P_1 x \quad P_2 x \quad \neg P_3 x}{f(x) \xrightarrow{c} x + y_1}$$

Given a non-smooth operation f, identifying a smooth function f' that captures the behaviour of f implies "smoothening" all the rules of f. For the rule ρ given above, a convenient smooth rule ρ' is:

$$\rho' = \frac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{b} y_2 \quad x_3 \overset{d}{\nrightarrow} \quad P_1 x_4 \quad P_2 x_5 \quad \neg P_3 x_6}{f'(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \xrightarrow{c} x_7 + y_1}$$

Note that the equation $f(x) = f'(x, x, x, x, x, x, x)$ holds and allows us to relate the behaviour of f with the one of its smoothened version.

Let G be an *preg* system and f a non-smooth l-ary function for G. The smoothening mechanism consists of the following steps (similar to those in [6]):

1. *Determine l' – the arity of f', the smooth version of f.* We start by counting how many fresh variables are needed for each position i in $\{1, \ldots, l\}$.

For each *preg* rule ρ for f (matching the format in Definition 3.2.1) and each i in $\{1,\ldots,l\}$, we first define a "barb" value $B(\rho,i)$, as follows:

$$B(\rho,i) = |I_i^+| + |J_i^+| + C_1 + C_2$$

where

$$C_1 = \begin{cases} 0, & \text{if } \mathcal{B}_i = \mathcal{Q}_i = \emptyset \\ 1, & \text{if } (\mathcal{B}_i = \emptyset \text{ and } \mathcal{Q}_i \neq \emptyset) \text{ or } (\mathcal{B}_i \neq \emptyset \text{ and } \mathcal{Q}_i = \emptyset) \\ 2, & \text{if } \mathcal{B}_i \neq \emptyset \text{ and } \mathcal{Q}_i \neq \emptyset \end{cases}$$

and

$$C_2 = \begin{cases} 0, & \text{if } x_i \text{ is not in the target of the conclusion} \\ 1, & \text{if } x_i \text{ is in the target of the conclusion} \end{cases}$$

Note that if ρ is smooth, then $(\forall i \in \{1,\ldots,l\})\,.\,B(\rho,i) \leq 1$. Symmetrically, if ρ is non-smooth, then $(\exists i \in \{1,\ldots,l\})\,.\,B(\rho,i) > 1$.

The number of variables needed for each $i \in \{1,\ldots,l\})$ is
$B(f,i) = max\{B(\rho,i) \mid \rho \text{ is a rule for } f\}$.

We conclude that the arity of the function f' is $l' = \sum_{i=\overline{1,l}} B(f,i)$.

2. *Determine the smooth rules ρ' for f', based on the rules ρ for f.* Consider ρ a rule for f, $\vec{w} = w_{11},\ldots,w_{1B(f,1)},\ldots,w_{l1},\ldots,w_{lB(f,l)}$ a vector of l' different fresh variables (that do not occur in ρ), and consider a substitution $\tau[w_{ij} \mapsto x_i]$ mapping each w_{ij} to x_i, $i \in \{1,\ldots,l\}$, $j \in \{1,\ldots,n_i\}$. Let ρ' be a smooth *preg* rule with the principal operation f', such that with the exception of their sources (resp. tests, for the case of predicate rules) $f(\vec{x})$ and $f'(\vec{w})$, $\rho'\tau[w_{ij} \mapsto x_i]$ and ρ are identical (note that this rule always exists). Then ρ' is the smooth version of f.

3. *Show that f and f' have the same behaviour.* This result is a consequence of the following lemma (similar to Lemma 4.12 in [6]):
Lemma 3.E.1. *Suppose G is a* preg *system, and $S = f(\vec{z})$ and $S' = f'(\vec{v})$ are terms in $\mathbb{T}(\Sigma_G)$ with variables that do not occur in $\mathcal{R}_G$. Suppose that there exists a 1-1 correspondence between rules for f and rules for f' such that, whenever a rule ρ for f is related to a rule ρ' for f', we have that $\rho\langle\vec{z}/\vec{x}\rangle$ and $\rho'\langle\vec{v}/\vec{y}\rangle$ are identical with exception of:*

   - *their sources $f(\vec{x})$ and, respectively, $f'(\vec{y})$, if ρ and ρ' are transition rules*

- *their tests $f(\vec{x})$ and, respectively, $f'(\vec{y})$, if ρ and ρ' are predicate rules*

Then

$$f(\vec{z}) = f'(\vec{v})$$

is sound for bisimilarity on $T(\Sigma_G)$.

The proof is a slightly modified version of the proof in [6]:

Proof. Consider $G' \sqsupseteq G$ and σ a closed $\Sigma_{G'}$–substitution. We have to prove that $s = f(\vec{z})\sigma \sim f'(\vec{v})\sigma = s'$, *i.e.*:

(a) $(\forall P \in \mathcal{P})\,.\, Ps \Leftrightarrow Ps'$.

(b) $(\forall a \in \mathcal{A})\,.\, s \xrightarrow{a} t \Leftrightarrow s' \xrightarrow{a} t$;

Case (a) "⇒". Assume Ps. We prove that Ps' holds.
As $\ltimes_{G'}, \rightarrow_{G'}$ are supported by G', $\mathcal{R}_{G'}$ contains a rule $\rho = \frac{H}{P(f(\vec{x}))}$ and there exists a $\Sigma_{G'}$–substitution τ s.t.:

(1) $\ltimes_{G'}, \rightarrow_{G'}, \tau \models H$

(2) $f(\vec{x})\tau = s$

We know that G' is a disjoint extension of G, therefore ρ is also a rule of G. So, there is $\rho' = \frac{H'}{P(f'(\vec{y}))}$ a rule of $\mathcal{R}_G \subseteq \mathcal{R}_{G'}$ s.t. $H\langle\vec{z}/\vec{x}\rangle = H'\langle\vec{v}/\vec{y}\rangle$ (*).

As in [6], Lemma 4.12, we consider τ' a $\Sigma_{G'}$–substitution defined as follows:

$$\tau'(w) = \begin{cases} \sigma(w) & \text{if } w \text{ occurs in } \vec{z} \text{ or } \vec{v} \\ \tau(w) & \text{otherwise} \end{cases}$$

and claim that for all variables w that do not occur in $\vec{z}$ or $\vec{v}$, $(\tau' \circ \langle\vec{z}/\vec{x}\rangle)(w) = \tau(w)$. At this point we infer:

(1) $\Rightarrow \ltimes_{G'}, \rightarrow_{G'}, \tau \models H \Rightarrow$(variables of H do not occur in $\vec{z}$ or $\vec{v}$)
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \circ \langle\vec{z}/\vec{x}\rangle \models H \Rightarrow$(general property of $\models$)
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \models H\langle\vec{z}/\vec{x}\rangle \Rightarrow$(by (*))
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \models H'\langle\vec{v}/\vec{y}\rangle \Rightarrow$(general property of $\models$)
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \circ \langle\vec{v}/\vec{y}\rangle \models H' \Rightarrow$(soundness of $\ltimes_{G'}$ for ρ')
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \circ \langle\vec{v}/\vec{y}\rangle \models P(f'(\vec{y})) \Rightarrow$
$\ltimes_{G'}, \rightarrow_{G'}, \tau' \models P(f'(\vec{v})) \Rightarrow$(def. $\models$)
$(P, f'(\vec{v})\tau') \in \ltimes_{G'} \Rightarrow$($\tau' = \sigma$ on $\vec{v}$)

$(P, f'(\vec{v})\sigma) \in \ltimes_{G'}$, so Ps' also holds.

For the case (b) "$\Rightarrow$", the reasoning is the same as in [6], Lemma 4.12.

Cases (a) "$\Leftarrow$" and (b) "$\Leftarrow$" are symmetric. □

The following lemma is a straightforward result of the smoothening procedure presented in steps (1.)–(3.):

Lemma 3.E.2. *Consider G a* preg *system and f a non-smooth l-ary operation for G. Then there exists $G' \sqsupseteq G$ with f' a smooth l'-ary operation for some l', and there exist two vectors of variables $\vec{z}$ and $\vec{v}$ such that*

$$f(\vec{z}) = f'(\vec{v})$$

is sound for bisimilarity on $T(\Sigma_{G''})$, for all the preg *systems G'' disjointly extending G'.*

Transforming smooth to smooth and distinctive operations. Given a smooth, but not distinctive, operation f', we next identify a family of smooth and distinctive operations $f'_1, \ldots, f'_n$ that "capture the behaviour of f'".

The procedure is identical to the one described in [6], Section 4.1.5. We consider $\mathcal{R}^1_G, \ldots, \mathcal{R}^n_G$, a partitioning of the set of rules for f' such that for all $i \in \{1, \ldots, n\}$, f' is distinctive in the *preg* system $(\Sigma_G, \mathcal{R}_G \setminus \bigcup_{\substack{j=\overline{1,n} \\ j \neq i}} \mathcal{R}^j_G)$. Note that this partitioning always exists; in the worst case each partition would be a singleton set.

Consider $\Sigma'_G = \Sigma_G \cup \{f'_1, \ldots, f'_n\}$, where $f'_1, \ldots, f'_n$ are fresh l'-ary operation symbols. Consider $\mathcal{R}'_G = \mathcal{R}_G \cup \{\widetilde{\mathcal{R}}^i_G \mid i \in \{1, \ldots, n\}\}$, where $\widetilde{\mathcal{R}}^i_G$ is obtained from $\mathcal{R}^i_G$ by replacing f' in the source with f'_i. Let $G' = (\Sigma'_G, \mathcal{R}'_G)$. Note that G' is a disjoint extension of G. It is trivial to check that

$$f'(\vec{x}) = f'_1(\vec{x}) + \ldots + f'_n(\vec{x})$$

is sound for bisimilarity on $T(\Sigma'_G)$.

Lemma 3.E.3. *Consider G a* preg *system and f a smooth l-ary operation for G. Then there exist $G' \sqsupseteq G$ and $f_1, \ldots, f_n$ smooth and distinctive l-ary operations for G' such that*

$$f(\vec{x}) = f_1(\vec{x}) + \ldots + f_n(\vec{x})$$

is sound for bisimilarity on $T(\Sigma_{G''})$, for all the preg *systems G'' disjointly extending G'.*

3.F A possible approach to handle implicit predicates

So far the current framework handles implicit predicates up to trees (*i.e.*, terms over the grammar (3.1)). For the case of arbitrary *preg* operations f we encountered some problems when proving the soundness of the action law $f(\vec{X}) = c.C[\vec{X}, \vec{y}]$ used in the head-normalization process.

Consider the closed instantiation $\vec{X} = \vec{s}$, $\vec{y} = \vec{t}$ and assume that $P(f(\vec{s}))$ holds for some predicate P in $\mathcal{P}^{\mathcal{I}}$. Then $P(c.C[\vec{s}, \vec{t}])$ should also hold, so we should have $P(C[\vec{s}, \vec{t}])$. One possible way of ensuring this property syntactically would be to stipulate some consistency requirements.

Consider, for instance, a scenario in which a transition rule for f has the conclusion $f(\vec{X}) \xrightarrow{c} C[\vec{X}, \vec{y}]$ and there is a predicate rule $\frac{H}{P(f[\vec{X}])}$. If there is a derivable predicate rule (ruloid) $\frac{H'}{P(C[\vec{X},\vec{y}])}$ with $H' \subseteq H$ then this would ensure that if the left-hand side of the action law satisfies P then so does then right-hand side. A symmetrical consistency requirement can be formulated to ensure that $P(f(\vec{s}))$ holds whenever $P(c.C[\vec{s}, \vec{t}])$ holds.

Note that for certain restricted *preg* systems these consistency requirements are guaranteed by the format of the rules. This is the case of the *preg* transition rules with "CCS-like" conclusions, *i.e.*, $C[\vec{X}, \vec{y}]$ is a variable or it is of the form $g(z_1, ..., z_m)$.

3.G Proof of Theorem 3.4.7

Proving [soundness]

Proof. Let $f \in \Sigma_G$ be a smooth and distinctive operation with the arguments $\vec{t} = t_1, \ldots, t_l$ as closed terms over Σ_G.

Distributivity law.

Let $i \in L$ be a positive position for f (every rule for f tests i positively) with $t_i = t_i' + t_i''$. We want to argue that the distributivity axiom for f is sound for position i.

Assume $\exists \rho \in \mathcal{R}^{\mathcal{A}}$ (of the form (1)) such that $f(t_1, \ldots, t_i' + t_i'', \ldots, t_l) \xrightarrow{c} t$ for some t. Then there are an action rule ρ and a closed substitution σ such that (a) σ satisfies the premises of ρ and (b) $f(t_1, \ldots, t_i' + t_i'', \ldots, t_l) \xrightarrow{c} t$ is obtained by instantiating the conclusion of ρ with σ. Since f is smooth and distinctive, i is a positive position for f, and ρ is a rule for f, we infer that there is a positive premise for i in ρ, i.e. $i \in I_\rho^+ \cup J_\rho^+$. In either case, we may assume, without loss of generality, that the positive premise is satisfied by t_i'. Take $\sigma' = \sigma[x_i \mapsto t_i']$. Then σ' satisfies the premises of ρ and proves that $f(t_1, \ldots, t_i', \ldots, t_l) \xrightarrow{c} t$. This is where we actually make use of the restriction for x_i to not occur in the target of the conclusion of ρ (Definition 3.4.1). Therefore, the right hand side of the instance of the distributivity law also has the transition to t that performs c.

We follow a similar reasoning for $f(t_1, \ldots, t_i', \ldots, t_l) + f(t_1, \ldots, t_i'', \ldots, t_l) \xrightarrow{c} t$.

The case in which $\exists \rho \in \mathcal{R}^{\mathcal{P}}$ (of the format (2)) such that $P(f(t_1, \ldots, t_i' + t_i'', \ldots, t_l))$ is also handled similarly (it is actually easier).

We conclude that

$$f(t_1, \ldots, t_i' + t_i'', \ldots, t_l) \sim f(t_1, \ldots, t_i', \ldots, t_l) + f(t_1, \ldots, t_i'', \ldots, t_l).$$

Action law. Let ρ be a transition rule (of the format (1)) for f s.t. $t_1, \ldots, t_l$ match the restrictions the over the arguments of f in Definition 3.4.5.2, and t_c be the corresponding closed instantiation of $C[\vec{X}, \vec{y}]$.

Obviously $c.t_c$ can only perform action c and reach t_c, and $c.t_c$ does not satisfy any predicate. By the hypothesis we know that ρ is a rule of f, so f can also perform c and reach the same t_c.

Next we want to prove that ρ is the only rule that applies for f (*i.e.*, $f(\vec{t})$ cannot perform any other action and does not satisfy predicates). Assume there exits $\rho' \neq \rho$ for f, that can be applied. As f is smooth and distinctive it follows that one of the following holds:

1. $\exists i \in I_\rho^+ \cap I_{\rho'}^+ \neq \emptyset$ s.t. $a_i \neq a_i'$ – case in which ρ' cannot be applied, since $t_i = a_i.t_i'$ can only perform action a_i;

2. $\exists i \in J_\rho^+ \cap J_{\rho'}^+ \neq \emptyset$ s.t. $P_i \neq P_i'$ – case in which ρ' cannot be applied, since $t_i = \kappa_{P_i}$ only satisfies P_i;

3. $\exists i \in I_\rho^+ \cap J_{\rho'}^+ \neq \emptyset$ – but this cannot be the case since $t_i = a_i.t_i'$ does not satisfy any predicate;

4. $\exists i \in J^+_\rho \cap I^+_{\rho'} \neq \emptyset$ – but this cannot be the case since $t_i = \kappa_{P_i}$ does not perform any action.

We thus reached a contradiction.

We conclude that

$$f(\vec{t}) \sim c.t_c \ .$$

Predicate law. Let ρ be a predicate rule (of the format (2)) for f s.t. $t_1, \ldots, t_l$ match the restrictions over the arguments of f in Definition 3.4.5.2.

Obviously that κ_P does not perform any action, and only satisfies P. By following a similar reasoning to the one for the case of *action laws*, we show that there is no rule $\rho' \neq \rho$ that applies for f. So, $f(\vec{t})$ does not perform any action, and does not satisfy any predicate besides P.

We conclude that

$$f(\vec{X}) \sim \kappa_P.$$

Deadlock law. The soundness follows immediately from the restrictions imposed by Definition 3.4.5.3. □

Proving head normalization

Proof. The result is an immediate consequence of the following claim:

Claim Let $f \in \Sigma$ be a smooth and distinctive operation with the arguments $\vec{t} = t_1, \ldots, t_l$ as closed terms over Σ_G in head normal form. Then there exists a closed term t over Σ_G in head normal form such that $E_G \vdash f(\vec{t}) = t$.

The proof follows by induction on the combined size of $t_1, \ldots, t_l$. We perform a case analysis:

1. $(\exists i \in L$ positive for $f) \ . \ t_i = t'_i + t''_i$. We apply the distributivity law and the inductive hypothesis.

2. $(\exists i \in L$ positive for $f) \ . \ t_i = \delta$. We apply a deadlock law.

3. $(\forall i \in L$ positive for $f) \ . \ t_i$ is either of the form $a_i.t'_i$ or κ_{P_i}.

   (a) $(\forall \rho \in \mathcal{R}$ for $f) \ . \ ((\exists j \in I^+_\rho) \ . \ t_j = a'_j.t'_j (a'_j \neq a_j)$ or $t_j = \kappa_{P'_j})$ or $((\exists j \in J^+_\rho) \ . \ t_j = a'_j.t'_j$ or $t_j = \kappa_{P'_j} (P'_j \neq P_j)) \Longrightarrow$ we can apply a deadlock law.

(b) $((\exists \rho \in \mathcal{R} \text{ for } f) \,.\, ((\forall j \in I^+_\rho) \,.\, t_j = a_j.t'_j)$ and $((\forall j \in J^+_\rho) \,.\, t_j = \kappa_{P_j})$.

Remark that, by the distinctiveness of f, ρ is the only rule that could be fired ($\bullet$). In order to prove this, assume, for instance, $\exists \rho' \neq \rho$ that could applied. If $I^+_\rho = J^+_\rho = \emptyset$ then this comes in contradiction with the second item of Definition 3.4.3. On the other hand, if $I^+_\rho \neq \emptyset$ then one of the following holds:

- $(\exists k \in I^+_\rho \cap I^+_{\rho'} \neq \emptyset) \,.\, a_k \neq a'_k$ – case in which ρ' cannot be applied, since t_k can only fire a_k;
- $\exists k \in I^+_\rho \cap J^+_{\rho'} \neq \emptyset$ – case in which we reach a contradiction, since by hypothesis we know that $(\forall j \in I^+_\rho) \,.\, t_j = a_j.t'_j$ which does not satisfy any predicate.

The reasoning is similar for $J^+_\rho \neq \emptyset$. So ρ is the only rule that can be applied for f.

We further identify the following situations:

i. $(\exists k \in I^-_\rho) \,.\, t_k = b.t'_k + t''_k \,(b \in \mathcal{B})$ or $(\exists k \in J^-_\rho) \,.\, t_k = t'_k + \kappa_Q \,(Q \in \mathcal{Q}_k) \overset{(\bullet)}{\Longrightarrow}$ we can apply a deadlock law.

ii. $(\forall k \in I^-_\rho \cup J^-_\rho) \,.\, t_k = \sum_{i \in I} a^i_k.t^i_k + \sum_{j \in J} \kappa_{P^j_k}$ s.t. $\{a^i_k \mid i \in I\} \cap \mathcal{B}_k = \emptyset$ and $\{P^j_k \mid j \in J\} \cap \mathcal{Q}_k = \emptyset$.

Recall the operator ∂ is used to specify restrictions on the actions performed and/or on the predicates satisfied by a given term. Therefore, for a term t_k satisfying the conditions in the hypothesis, it is intuitive to see that $t_k = \partial_{\mathcal{B},\mathcal{Q}}(t_k)$. In what follows, we provide an auxiliary result formalizing this intuition:

Lemma 3.G.1. *Consider $G \sqsupseteq FTP^\partial$ a* preg *system and a term $t = \sum_{i \in I} a_i.t_i + \sum_{j \in J} \kappa_{P_j} \in T(\Sigma_G)$ in h.n.f. Let $\mathcal{B}$ and $\mathcal{Q}$ be two sets of restricted actions and predicates, respectively. If $\{a_i \mid i \in I\} \cap \mathcal{B} = \emptyset$ and $\{P_j \mid j \in J\} \cap \mathcal{Q} = \emptyset$ then $E^\partial_{FTP} \vdash t = \partial_{\mathcal{B},\mathcal{Q}}(t)$.*

Proof. The proof follows immediately, by induction on the structure of t and by applying the axioms (A_6) – (A_{12}). □

In these conditions, according to Lemma 3.G.1, it holds that $(\forall k \in I^-_\rho \cup J^-_\rho) \,.\, t_k = \partial_{\mathcal{B}_k,\mathcal{Q}_k}(t_k)$. Therefore, all the requirements for applying

a trigger law are met: if $\rho \in \mathcal{R}^{\mathcal{A}}$ then apply the action law for ρ, and if $\rho \in \mathcal{R}^{\mathcal{P}}$ then apply the predicate law for ρ.

This reasoning suffices to guarantee that E_G is head normalizing. □

3.H Proof of Lemma 3.5.3

Proof. Let $t, t' \in T(\Sigma_{FTP_/})$ s.t. $(\forall n \in \mathbb{N}) \,.\, t/c^n = t'/c^n$. We have $(\forall n \in \mathbb{N}) \,.\, t/c^n \sim t'/c^n$ and we want to prove that $t \sim t'$. In other words, we have to build a bisimulation relation $R \subseteq T(\Sigma_{FTP_/}) \times T(\Sigma_{FTP_/})$ s.t. $(t, t') \in R$.

First we make the following observations:

1. $\forall t \in T(\Sigma_{FTP_/})$ it holds that t is finitely branching (by Lemma 3.2.9);
2. if $t/c^n \sim t'/c^n$ then $(\forall P \in \mathcal{P}) \,.\, (Pt \Leftrightarrow Pt')$, where $t, t' \in T(\Sigma_{FTP_/})$ and $n \in \mathbb{N}$ (the reasoning is as follows: if $t/c^n \sim t'/c^n$ then by the definition of bisimilarity it holds that $P(t/c^n) \Leftrightarrow P(t'/c^n)$, so by (rl_{11}) it follows that $Pt \Leftrightarrow Pt'$).

Next we show that the relation $R \subseteq T(\Sigma_{FTP_/}) \times T(\Sigma_{FTP_/})$ defined as

$$(t, t') \in R \text{ iff } (\forall n \in \mathbb{N}) \,.\, t/c^n \sim t'/c^n$$

is a bisimulation relation.

Assume $(t, t') \in R$ and $t \xrightarrow{a} t_1$ and for $n > 0$ define $S_n = \{t^* \mid t' \xrightarrow{a} t^* \text{ and } t_1/c^n \sim t^*/c^n\}$. Note that:

- $S_1 \supseteq S_2 \supseteq \ldots$, as $t_1/c^{n+1} \sim t^*/c^{n+1} \Rightarrow t_1/c^n \sim t^*/c^n$ (the latter implication is straightforward by the definition of bisimilarity and by (rl_{10}), (rl_{11}));
- $(\forall n > 0) \,.\, S_n \neq \emptyset$, since by the definition of R we have that $t_1/c^{n+1} \sim t^*/c^{n+1}$, and $t \xrightarrow{a} t_1$ according to the hypothesis;
- each S_n is finite, as t' is finitely branching (1.).

At this point we conclude that the sequence $S_1, S_2, \ldots, S_n, \ldots$ remains constant from some n onwards. Therefore $\cap_{n>0} S_n \neq \emptyset$. Let t' be an element of this intersection. We have that: $t' \xrightarrow{a} t'_1$, $(t_1, t'_1) \in R$ and $(\forall P \in \mathcal{P}) \,.\, Pt \Rightarrow_{(2.)} Pt'$ (♣). We follow a similar reasoning for the symmetric case $t' \xrightarrow{a} t'_1$ and show that $\exists t_1$ s.t. $t \xrightarrow{a} t_1$, for $(t_1, t'_1) \in R$ and $(\forall P \in \mathcal{P}) \,.\, Pt' \Rightarrow_{(2.)} Pt$ (♠). By (♣) and (♠) it follows that R is a bisimulation relation, so $(t, t') \in R \Rightarrow t \sim t'$. □

3.I A thorough analysis on GSOS with Predicates

In this section we provide a classification of the predicates a language designer can operate with, introducing new types of predicates besides those already presented in this chapter. We then provide an adapted rule format that includes these predicates, as well as an axiom schema that is sound and ground-complete modulo bisimilarity.

3.I.1 Predicate classification

Recall that we work with a finite set of actions $\mathcal{A}$ and with the standard signature for finite trees Σ_{FT}, consisting of δ, the deadlock operation, $\_ + \_$, the nondeterministic choice, and $a.\_$, the action prefix with $a \in \mathcal{A}$. A finite tree term t is built according to the following grammar

$$t ::= \delta \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t \ .$$

We want to be able to reason on process predicate satisfiability in a syntactic manner. The idea is to denote that a closed term t satisfies (or does not satisfy) a predicate by means of witnessing subterms.

To get the intuition for this idea consider four properties. A process may immediately terminate ($\downarrow$), eventually terminate ($\lightning$), perform action a ($\stackrel{a}{\rightarrow}$)[1], or it may not perform action a at all ($\stackrel{a}{\not\rightarrow}$). In order to be able check for the first two properties in a syntactic manner we need to consider two corresponding constant processes, $\kappa_{\downarrow}$ and $\kappa_{\lightning}$, such that $\kappa_{\downarrow}$ satisfies $\downarrow$ and $\kappa_{\lightning}$ satisfies $\lightning$.

We now have enough information to reason, for instance, on the properties satisfied by the process denoted by the term $a.a.\delta + a.\kappa_{\lightning} + \kappa_{\downarrow}$. It can immediately terminate because it has $\kappa_{\downarrow}$ as a top witnessing constant summand. It can eventually terminate because it has $\kappa_{\lightning}$ as a witnessing subterm, which, in this case, does not have to be a top summand. It satisfies $\stackrel{a}{\rightarrow}$ because it has $a.t$ ($t \equiv a.\delta$) as a summand subterm. At the same time $a.t$ acts as a negative witness for the property $\stackrel{a}{\not\rightarrow}$.

[1] It is convenient to denote the property *can perform action a* by the construct "$\stackrel{a}{\rightarrow}$", which is the same one used when building a positive transition formula. One can always distinguish between the transition formula and the predicate formula, as the former involves writing a term on the right hand side of the construct (*e.g.* "$x \stackrel{a}{\rightarrow} y$"), while the latter does not (*e.g.* "$x \stackrel{a}{\rightarrow}$").

On the other hand $\kappa_{\downarrow} + \kappa_{\lightning}$ does satisfy $\overset{a}{\nrightarrow}$ as it does not have a negative witness subterm of the form $a.t'$ as a summand.

Let us now make a thorough analysis on the possible classes of predicates. We denote by $\mathcal{P}$ the set of all considered predicates.

Existential/Universal predicates $\mathcal{P}$ is partitioned into *existential* and *universal* predicates, subsets denoted by $\mathcal{P}_{\exists}$ and $\mathcal{P}_{\forall}$, respectively. Intuitively, a process satisfies an existential predicate when there is at least one execution path on which the predicate is satisfied, and a universal predicate when it is satisfied on all possible execution paths. The formal distinction between existential and universal predicates is given by the requirement that a term of the form $s + t$ satisfies an existential predicate whenever s or t satisfies it, and it satisfies a universal predicate whenever both s and t satisfy it. For example, $\downarrow$, $\lightning$ and $\overset{a}{\rightarrow}$ are existential predicates, while $\overset{a}{\nrightarrow}$ is universal.

When it is the case that an existential predicate P represents the negation of a universal predicate Q (*e.g.* $\overset{a}{\rightarrow}$ is the negation of $\overset{a}{\nrightarrow}$), we denote this by $P = \overline{Q}$, or $Q = \overline{P}$. Here, $\overline{\cdot} : \mathcal{P} \to \mathcal{P}$ is a partially defined bijective mapping between existential and universal predicates.

Important convention From this point forward we will assume, for the ease of presentation, that if a predicate P is in $\mathcal{P}$, then its negation $\overline{P}$ is also in $\mathcal{P}$ ($\mathcal{P}$ is closed under the application of $\overline{\cdot}$). This convention allows us to consider a single class of witnessing constants that act both as positive witnesses for existential predicates and negative witnesses for universal predicates.

Remark 3.I.1. *The deadlock constant δ satisfies all universal predicates. At the same time, any constant κ_P, for some existential predicate P, satisfies a universal predicate Q as long as $P \neq \overline{Q}$. Also, any closed term t of the form $a.t'$ satisfies the universal predicate Q as long as t is not a negative witness for Q.*

This intuition is later formalized by the operational semantics of predicates presented in Section 3.I.3.

Direct predicates We denote by $\mathcal{P}^{\mathcal{D}}$ the class of predicates directly satisfied by the terms of the form $a.t'$ (for some action a) . Predicates in this class are named *direct predicates*. We denote by $\mathcal{A}_P^{\mathcal{D}} \subseteq \mathcal{A}$ the set consisting of all the actions for which this behaviour is permitted when reasoning on the satisfiability of the

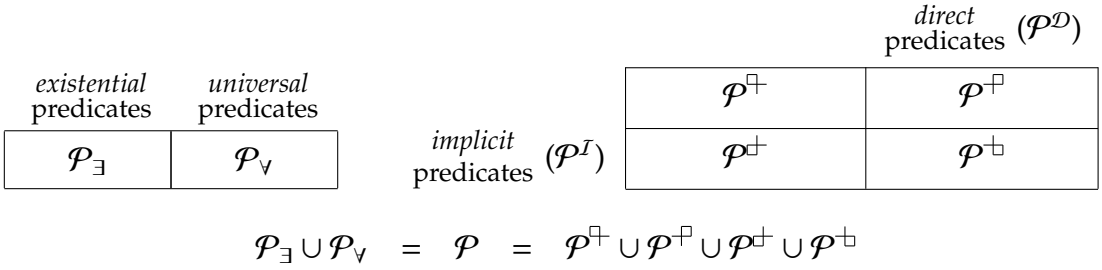
direct predicate $P \in \mathcal{P}^{\mathcal{D}}$. We refer to $\mathcal{A}_P^{\mathcal{D}}$ as the set of *supported direct actions* for predicate P. In the context of the set of actions $\mathcal{A} = \{a, b, c\}$, both $\xrightarrow{a}$ and $\overset{a}{\nrightarrow}$ are direct predicates, but $\mathcal{A}^{\mathcal{D}}_{\xrightarrow{a}} = \{a\}$, while $\mathcal{A}^{\mathcal{D}}_{\overset{a}{\nrightarrow}} = \{b, c\}$. By convention, $\mathcal{A}_P^{\mathcal{D}} = \emptyset$ for any predicate P that is not direct.

Implicit predicates Another class we consider is that of predicates who hold implicitly for terms of the form $a.t$ (for some actions a) whenever they hold for t. We denote this class by $\mathcal{P}^{\mathcal{I}}$, which consists of what we call *implicit predicates*. An example of an implicit predicate is the eventual termination $\lightning$. We denote by $\mathcal{A}_P^{\mathcal{I}} \subseteq \mathcal{A}$ the set of all the actions which imply this behaviour when reasoning on the satisfiability of the implicit predicate $P \in \mathcal{P}^{\mathcal{I}}$. We refer to $\mathcal{A}_P^{\mathcal{I}}$ as the set of *supported implicit actions* for predicate P. For the case of the predicate $\lightning$, it is always the case that $\mathcal{A}_P^{\mathcal{I}} = \mathcal{A}$. By convention, $\mathcal{A}_P^{\mathcal{I}} = \emptyset$ for any predicate P that is not implicit.

Remark 3.I.2. *There exist predicates that are neither direct, nor implicit, such as immediate successful termination $\downarrow$. On the other side of the spectrum, there also exist predicates that are both direct and implicit. Consider, for instance the existential property according to which a process can eventually perform action a, denoted by $\lightning_a$. Obviously $a.t \lightning_a$ holds, so the predicate is direct, and $b.t \lightning_a$ holds for some $b \neq a$ whenever $t \lightning_a$ holds, which means it is also implicit.*

It is worth noting that the property "can never perform action a eventually" (the negation of $\lightning_a$, denoted by $\not\lightning_a$) is universal and implicit, but not direct. This is because we cannot directly infer that $b.t \not\lightning_a$ holds for some action b. The property is verified only when $b \neq a$ and $t \not\lightning_a$ holds.

Figure 3.8 presents the overview of the predicate classes. As it can be seen, the set of direct predicates $\mathcal{P}^{\mathcal{D}}$ consists of direct predicates that are implicit ($\mathcal{P}^{\boxplus}$) and direct predicates that are not implicit ($\mathcal{P}^{\boxplus}$). Therefore $\mathcal{P}^{\mathcal{D}} = \mathcal{P}^{\boxplus} \cup \mathcal{P}^{\boxplus}$, which we also denote by $\mathcal{P}^{\boxplus}$. Similarly, $\mathcal{P}^{\mathcal{I}} = \mathcal{P}^{\boxplus} \cup \mathcal{P}^{\boxplus} = \mathcal{P}^{\boxplus}$.



$$\mathcal{P}_\exists \cup \mathcal{P}_\forall \quad = \quad \mathcal{P} \quad = \quad \mathcal{P}^{\boxplus} \cup \mathcal{P}^{\boxplus} \cup \mathcal{P}^{\boxplus} \cup \mathcal{P}^{\boxplus}$$

Figure 3.8: Predicates classification

In order to familiarize the reader with the notations, we present the class each of the predicates mentioned so far is in.

$$\downarrow \in \mathcal{P}_{\exists}^{\sqcap\sqcap} \qquad \xrightarrow{a} \in \mathcal{P}_{\exists}^{\dashv\sqcap} \qquad \lightning_a \in \mathcal{P}_{\exists}^{\dashv\sqcup}$$
$$\lightning \in \mathcal{P}_{\exists}^{\sqcup\sqcup} \qquad \overset{a}{\nrightarrow} \in \mathcal{P}_{\forall}^{\dashv\sqcap} \qquad \overset{}{\not\lightning}_a \in \mathcal{P}_{\forall}^{\dashv\sqcup}$$

Figures 3.9 and 3.10 include many examples of existential and, respectively, universal predicates, in the context of the set of actions $\mathcal{A} = \{a, b, c\}$.

$\mathcal{P}_{\exists}$		$\mathcal{P}^{\mathcal{D}}$
	$\downarrow$ can immediately successfully terminate	$\overrightarrow{\mathcal{A}}^{\mathcal{D}}_{\xrightarrow{a}} = \{a\}$ can immediately perform a $\overrightarrow{\mathcal{A}}^{\mathcal{D}}_{\rightarrow} = \{a, b, c\}$ can immediately perform an action $\overrightarrow{\mathcal{A}}^{\mathcal{D}}_{\xrightarrow{\not a}} = \{b, c\}$ can immediately avoid performing a
$\mathcal{P}^{\mathcal{I}}$	$\lightning$ can eventually successfully terminate $\mathcal{A}^{\mathcal{I}}_{\lightning} = \{a, b, c\}$ $\lightning^{a}_{a}$ can take an "a-path" up to witnessing constant $\mathcal{A}^{\mathcal{I}}_{\lightning^{a}_{a}} = \{a\}$ $\lightning^{\not a}_{\not a}$ can avoid an "a-path" up to witnessing constant $\mathcal{A}^{\mathcal{I}}_{\lightning^{\not a}_{\not a}} = \{b, c\}$	$\lightning_a$ can eventually perform a on a path $\mathcal{A}^{\mathcal{D}}_{\lightning_a} = \{a\}$ $\mathcal{A}^{\mathcal{I}}_{\lightning_a} = \{b, c\}$ $\lightning_{\not a}$ can eventually avoid a on a path $\mathcal{A}^{\mathcal{D}}_{\lightning_{\not a}} = \{b, c\}$ $\mathcal{A}^{\mathcal{I}}_{\lightning_{\not a}} = \{a\}$

Figure 3.9: Examples of existential predicates

Remark 3.I.3. *If a predicate P is implicit, then its negation $\overline{P}$ is also implicit and, moreover, they have the same set of supported implicit actions. Formally, $P \in \mathcal{P}^{\sqcup\sqcup}$* iff *$\overline{P} \in \mathcal{P}^{\sqcup\sqcup}$ and $\mathcal{A}^{\mathcal{I}}_{P} = \mathcal{A}^{\mathcal{I}}_{\overline{P}}$.*

To understand the intuition behind this remark, consider a term t and an implicit existential predicate $P \in \mathcal{P}^{\sqcup\sqcup}_{\exists}$. In order to verify if t satisfies P one needs to check if t has a summand that witnesses P or a summand of the form $\alpha.t'$ for some action $\alpha \in \mathcal{A}^{\mathcal{I}}_{P}$ and term t' that satisfies P (which is recursively checked in the same manner as for t). On the other hand, in order to verify if t satisfies $\overline{P}$, one needs to make sure that t has no negative witnessing summand and that for every summand of the form $\alpha.t'$, with $\alpha \in \mathcal{A}^{\mathcal{I}}_{P}$ the same statement holds recursively for t'. Note that the summands for which $\alpha \notin \mathcal{A}^{\mathcal{I}}_{P}$ are

$\mathcal{P}_{\forall}$		$\mathcal{P}^{\mathcal{D}}$
	$\not\rightarrow$ cannot perform any action	$\overset{a}{\not\rightarrow}$ cannot immediately perform a $\mathcal{A}^{\mathcal{D}}_{\overset{a}{\not\rightarrow}} = \{b, c\}$ $\not\downarrow$ does not immediately terminate $\mathcal{A}^{\mathcal{D}}_{\not\downarrow} = \{a, b, c\}$ $\Uparrow_a$ can immediately only perform a $\mathcal{A}^{\mathcal{D}}_{\Uparrow_a} = \{a\}$
$\mathcal{P}^{\mathcal{I}}$	$\not\downarrow$ can never successfully terminate $\mathcal{A}^{\mathcal{I}}_{\not\downarrow} = \{a, b, c\}$ $\not\downarrow_a$ can never perform a on any path $\mathcal{A}^{\mathcal{I}}_{\not\downarrow_a} = \{b, c\}$ $\Uparrow_a$ can only perform a in the future $\mathcal{A}^{\mathcal{I}}_{\Uparrow_a} = \{a\}$	$\overline{\not\downarrow_a}$ can avoid performing a somewhere on every path before witnessing constant $\mathcal{A}^{\mathcal{D}}_{\overline{\not\downarrow_a}} = \{b, c\}$ $\mathcal{A}^{\mathcal{I}}_{\overline{\not\downarrow_a}} = \{a\}$ $\overline{\not\downarrow_{a'}}$ can perform a somewhere on every path before witnessing constant $\mathcal{A}^{\mathcal{D}}_{\overline{\not\downarrow_{a'}}} = \{a\}$ $\mathcal{A}^{\mathcal{I}}_{\overline{\not\downarrow_{a'}}} = \{b, c\}$

Figure 3.10: Examples of universal predicates

of no interest because they do not satisfy P by construction, so they satisfy $\overline{P}$. A similar reasoning is used if $P \in \mathcal{P}^{\boxplus}_{\forall}$.

It is reasonable to require that one is able to check for the satisfiability of a predicate P by $a.t$ either by using the direct or the implicit reasoning. At the same time, we want to avoid clashing predicate definitions:

- it cannot be the case that $\alpha.t$ satisfies both $\overset{a}{\rightarrow}$ and $\overset{a}{\not\rightarrow}$, no matter which action α stands for;
- it cannot be the case that $\alpha.t$ satisfies both $\not\downarrow_a$ and $\not\downarrow_a$ (the latter would require for $t \not\downarrow_a$ to hold), no matter which action α stands for.

We formalize these observations as language definition consistency requirements.

Consistency Requirement 3.I.4. *Suppose the language characterization includes both a predicate P and its negation $\overline{P}$. In this case,*

1. *if $P \in \mathcal{P}^{\boxplus}$ then $\mathcal{A}^{\mathcal{D}}_P = \mathcal{A} \setminus \mathcal{A}^{\mathcal{D}}_{\overline{P}}$, and*
2. *if $P \in \mathcal{P}^{\boxplus}$ then $\mathcal{A}^{\mathcal{D}}_P = \mathcal{A} \setminus \mathcal{A}^{\mathcal{I}}_{\overline{P}}$.*

3.I.2 The *preg*$^+$ rule format

Definition 3.I.5 (*preg*$^+$ rule format)**.** *Consider $\mathcal{A}$, a set of actions, and $\mathcal{P}$, a set of predicates. A preg*$^+$ transition / predicate rule *for an l-ary operation f is a deduction rule of the form:*

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I, j \in I_i\} \quad \{P_{ij}x_i \mid i \in J, j \in J_i\} \quad \{Q_{ij}x_i \mid i \in K, j \in K_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}] \;\; / \;\; P(f(x_1, \ldots, x_l))}$$

where

1. *$x_1, \ldots, x_l$, and y_{ij}'s are pairwise distinct variables,*
2. *$I, J, K \subseteq L = \{1, \ldots, l\}$ and each I_i, J_i and K_i is finite,*
3. *$a_{ij}, c \in \mathcal{A}$, and*
4. *$P_{ij} \in \mathcal{P}_\exists$, $Q_{ij} \in \mathcal{P}_\forall$, $P \in \mathcal{P}$.*

Note that this definition does not include negative premises. This is because negative transitions are regarded as universal predicates, and because every existential predicate has a universal one as negation and vice-versa.

Definition 3.I.6 (*preg*$^+$ system)**.** *A* preg$^+$ *system is a pair $G = (\Sigma_G, \mathcal{R}_G)$, where Σ_G is a finite signature and $\mathcal{R}_G = \mathcal{R}_G^{\mathcal{A}} \cup \mathcal{R}_G^{\mathcal{P}}$ is a finite set of* preg *rules over Σ_G ($\mathcal{R}_G^{\mathcal{A}}$ and $\mathcal{R}_G^{\mathcal{P}}$ represent the transition and, respectively, the predicate rules of G).*

Definition 3.I.7 (Bisimulation)**.** *Consider a* preg *system $G = (\Sigma_G, \mathcal{R}_G)$. A symmetric relation $R \subseteq T(\Sigma_G) \times T(\Sigma_G)$ is a* bisimulation *if, and only if:*

1. *for all $s, t, s' \in T(\Sigma_G)$, whenever $(s, t) \in R$ and $s \xrightarrow{a} s'$ for some $a \in \mathcal{A}$, then there is some $t' \in T(\Sigma_G)$ such that $t \xrightarrow{a} t'$ and $(s', t') \in R$;*
2. *whenever $(s, t) \in R$ and Ps ($P \in \mathcal{P}$) then Pt.*

Two closed terms s and t are bisimilar *(written $s \sim t$) if, and only if, there is a bisimulation relation R such that $(s, t) \in R$.*

3.I.3 Finite trees with predicates

We extend Σ_{FT} with a set of constants: $\Sigma_{FTP} = \Sigma_{FT} \cup \{\kappa_P \mid P \in \mathcal{P}_{\exists}^{\square\square}\}$. Recall that $P \in \mathcal{P}_\exists$ if, and only if, $\overline{P} \in \mathcal{P}_\forall$, and that κ_P is both a positive witness for P and a negative witness for $\overline{P}$.

Remark 3.I.8. *The predicates in $\mathcal{P}_{\exists}^{\sqcap}$, such as $\xrightarrow{a}$, need no witnessing constants because terms of the form $a.t$ serve as witnesses.*

One might think that the constants for predicates in $\mathcal{P}_{\exists}^{\sqcup}$ should be excluded, and the truth is that when specifying terms these constants should not be used. However, we include them because they provide a good mechanism for reasoning on term equivalence by means of axioms when considering implicit direct predicates.

The operational semantics of *FTP* is given by the *preg*$^+$ set of rules TSS_{FTP}:

$$\forall a \in \mathcal{A}: \quad \frac{}{a.x \xrightarrow{a} x}\,(rl_1) \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}\,(rl_2) \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}\,(rl_3)$$

$$\forall P \in \mathcal{P}_{\exists}: \quad \frac{Px}{P(x+y)}\,(rl_4) \qquad \frac{Py}{P(x+y)}\,(rl_5) \qquad \frac{}{P(\kappa_P)}\,(rl_6)$$

$$\forall P \in \mathcal{P}_{\forall}: \quad \frac{Px\ Py}{P(x+y)}\,(rl_7) \qquad \frac{}{P(\delta)}\,(rl_8) \qquad \frac{}{P(\kappa_Q)}\ \text{if } P \neq \overline{Q}\ (rl_9)$$

$$\begin{array}{l}\forall P \in \mathcal{P}^{\sqcap} \\ \forall a \in \mathcal{A}_P^{\mathcal{D}}\end{array}: \quad \frac{}{P(a.x)}\,(rl_{10}) \qquad \begin{array}{l}\forall P \in \mathcal{P}^{\sqcup} \\ \forall a \in \mathcal{A}_P^{\mathcal{I}}\end{array}: \quad \frac{Px}{P(a.x)}\,(rl_{11})$$

Figure 3.11: The semantics of finite trees with predicates TSS_{FTP}

The system E_{FTP} consists of the following axioms:

$$\begin{array}{ll}
(A_1) & x + y = y + x \\
(A_2) & (x + y) + z = x + (y + z) \\
(A_3) & x + x = x \\
(A_4) & x + \delta = x \\
(A_5) & a.x = a.x + \kappa_P, \quad \forall P \in \mathcal{P}_{\exists}^{\sqcup}\ \forall a \in \mathcal{A}_P^{\mathcal{D}} \\
 & \qquad\qquad\qquad\quad \forall \overline{P} \in \mathcal{P}_{\forall}^{\sqcup}\ \forall a \in \mathcal{A} \setminus \mathcal{A}_{\overline{P}}^{\mathcal{I}} \\
(A_6) & a.(x + \kappa_P) = a.(x + \kappa_P) + \kappa_P, \quad \forall P \in \mathcal{P}_{\exists}^{\sqcup}\ \forall a \in \mathcal{A}_P^{\mathcal{I}}
\end{array}$$

Figure 3.12: The axiom system E_{FTP}

Theorem 3.I.9. *E_{FTP} is sound and ground-complete for bisimilarity on $T(\Sigma_{FTP})$.*

Proof. It is easy to see that the axioms are head normalizing, in the sense given in Definition 3.3.1, which immediately helps inferring that the system is ground-complete.

Concerning soundness, it is sufficient to only focus on the axiom (A_5) as the soundness of the other axioms follows from Theorem 3.3.3.

We want to prove that $s \equiv a.p \sim t \equiv a.p + \kappa_P$ for a fixed a and P such that either $P \in \mathcal{P}^{\dotplus}_{\exists}, a \in \mathcal{A}^{\mathcal{D}}_P$ or $\overline{P} \in \mathcal{P}^{\dotplus}_{\forall}, a \in \mathcal{A} \setminus \mathcal{A}^{\mathcal{I}}_P$.

It is trivial to check that $s \xrightarrow{a} p$ and $t \xrightarrow{a} p$, by using rules $(rl_{1,2,3})$.

Let us now prove that both terms satisfy the same predicates.

"$\Rightarrow$" Consider $Q \in \mathcal{P}$ such that $Q(a.p)$ holds (♠). Let us prove that $Q(a.p + \kappa_P)$ also holds.

If $Q \in \mathcal{P}_{\exists}$ then $Q(a.p + \kappa_P)$ is inferred by using $(rl_{5,6})$.

If $Q \in \mathcal{P}_{\forall}$ then we need to show that $Q(\kappa_P)$ holds, and afterwards apply (rl_7).

1. $P \in \mathcal{P}^{\dotplus}_{\exists}$ and $a \in \mathcal{A}^{\mathcal{D}}_P$. In this case $Q(\kappa_P)$ holds only if $Q \neq \overline{P}$ (by rule (rl_9)). Here κ_P is the positive witness for P and negative witness for Q.

   Let us prove by *Reductio ad Absurdum* that $Q \neq \overline{P}$. Assume that $Q = \overline{P}$. As $P \in \mathcal{P}^{\dotplus}_{\exists}$ and $a \in \mathcal{A}^{\mathcal{D}}_P$, we know that $(rl_{10}) : \frac{}{\overline{P}(a.x)}$ is applicable. By Consistency Requirement 3.I.4, neither of these rules exist: $\frac{}{Q(a.x)}$, $\frac{Qx}{Q(a.x)}$. Therefore there is no way to infer that $Q(a.x)$ holds, which is false due to (♠).

2. $\overline{P} \in \mathcal{P}^{\dotplus}_{\forall}$ and $a \in \mathcal{A} \setminus \mathcal{A}^{\mathcal{I}}_{\overline{P}}$. In this case $Q(\kappa_P)$ holds only if $Q \neq \overline{P}$ (by rule (rl_9)). Here κ_P is the negative witness for $\overline{P}$ and Q.

   Let us prove by *Reductio ad Absurdum* that $Q \neq \overline{P}$. Assume that $Q = \overline{P}$. As $\overline{P} \in \mathcal{P}^{\dotplus}_{\forall}$ and $a \in \mathcal{A} \setminus \mathcal{A}^{\mathcal{I}}_P$, we know that $(rl_{11}) : \frac{\overline{P}x}{\overline{P}(a.x)}$ cannot be applied. Therefore $\frac{Qx}{Q(a.x)}$ cannot be applied. At the same time $\overline{P} \notin \mathcal{P}^{\dotplus\!\!\square}$, so there is no rule $\frac{}{\overline{P}(a.x)}$, which means that rule $\frac{}{Q(a.x)}$ does not exist. Therefore there is no way to infer that $Q(a.x)$ holds, which is false due to (♠).

"$\Leftarrow$" Consider that $Q(a.p + \kappa_P)$ holds. Let us prove that $Q(a.p)$ holds.

If $Q \in \mathcal{P}_{\forall}$, then the property is inferred by using (rl_7).

Suppose $Q \in \mathcal{P}_{\exists}$.

1. $P \in \mathcal{P}^{\dotplus}_{\exists}$ and $a \in \mathcal{A}^{\mathcal{D}}_P$. If $Q = P$ then $Q \in \mathcal{P}^{\dotplus\!\!\square}$, and as $a \in \mathcal{A}^{\mathcal{D}}_Q$, $(rl_{11}) : \frac{}{Q(a.x)}$ can be applied. If $Q \neq P$ then $Q(\kappa_P)$ does not hold, so $Q(a.p)$ must hold.

2. $\overline{P} \in \mathcal{P}^{\dotplus}_{\forall}$ and $a \in \mathcal{A} \setminus \mathcal{A}^{\mathcal{I}}_{\overline{P}}$. If $Q \neq P$ then $Q(\kappa_P)$ does not hold, therefore $Q(a.p)$ must hold. If $Q = P$ then by Consistency Requirement 3.I.4 it holds that $a \in \mathcal{A}^{\mathcal{D}}_{P=Q}$, so rule $(rl_{11}) : \frac{}{Q(a.x)}$ can be applied.

□

3.I.4 Axiomatizing arbitrary *preg*$^+$ operations

We denote the set of all rules for f by $\mathcal{R}_f = \mathcal{R}_f^{\mathcal{A}} \cup \mathcal{R}_f^{\mathcal{P}}$.

First we introduce a meta-operation that checks whether a head normal form argument of an operation satisfies the requirements of a rule for that operation.

$\checkmark : T(\Sigma) \times L \times \mathcal{R}_f \to \{true, false\}$

$$\checkmark\left(t, i, \frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I, j \in I_i\} \quad \{P_{ij}x_i \mid i \in J, j \in J_i\} \quad \{Q_{ij}x_i \mid i \in K, j \in K_i\}}{\phi}\right) =$$

$$\left(i \in I \Rightarrow \forall j \in I_i\ (\exists z \in T(\Sigma)\ (t \equiv a_{ij}.y_{ij} + z))\right)$$

$$\text{and} \begin{pmatrix} i \in J \Rightarrow \forall j \in J_i\ (\ (P_{ij} \in \mathcal{P}_{\exists}^{\boxplus} \text{ and } \exists z \in T(\Sigma)\ (t \equiv \kappa_{P_{ij}} + z)) \text{ or} \\ \qquad\qquad (P_{ij} \in \mathcal{P}_{\exists}^{\boxplus} \text{ and } \exists y, z \in T(\Sigma)\ (\exists a \in \mathcal{A}_{P_{ij}}^{\mathcal{D}}\ (t \equiv a.y + z)))\ ) \end{pmatrix}$$

$$\text{and} \begin{pmatrix} i \in K \Rightarrow \forall j \in K_i\ (\ (Q_{ij} \in \mathcal{Q}_{\forall}^{\boxplus} \text{ and } \forall z \in T(\Sigma)\ (t \not\equiv \kappa_{\overline{Q_{ij}}} + z)) \text{ or} \\ \qquad\qquad (Q_{ij} \in \mathcal{Q}_{\forall}^{\boxplus} \text{ and } \forall y, z \in T(\Sigma)\ (\forall a \in \mathcal{A}_{\overline{Q_{ij}}}^{\mathcal{D}}(t \not\equiv a.y + z)))\ ) \end{pmatrix}.$$

We extend $\checkmark$ so that it performs the same check for every argument of an operation:

$$\checkmark(\vec{t}, \rho) = \bigwedge_{t_i \in \vec{t}} (\checkmark(t_i, i, \rho))$$

Definition 3.I.10. *Consider a* preg$^+$ *system* $G = (\Sigma, L, \mathcal{R})$ *such that* $FTP \sqsubseteq G$. *By* E_G *we denote the axiom system that extends* E_{FTP} *with the following schema for every operation* f *in* Σ, *parameterized over the vector of process terms* $\vec{X}$ *in head normal form:*

$$f(\vec{X}) = \sum \left\{ c.C[\vec{X}, \vec{y}] \mid \forall \rho = \frac{H}{f(\vec{X}) \xrightarrow{c} C[\vec{X}, \vec{y}]} \in \mathcal{R}_f^{\mathcal{A}}\ (\checkmark(\vec{X}, \rho)) \right\}$$

$$+ \sum \left\{ k_P \mid \forall P \in \mathcal{P}_{\exists}^{\boxplus}\ (\exists \rho = \frac{H}{P(f(\vec{X}))} \in \mathcal{R}_f^{\mathcal{P}}\ (\checkmark(\vec{X}, \rho))) \right\}$$

$$+ \sum \left\{ k_{\overline{Q}} \mid \forall Q \in \mathcal{P}_{\forall}^{\boxplus}\ (\forall \rho = \frac{H}{Q(f(\vec{X}))} \in \mathcal{R}_f^{\mathcal{P}}\ (\neg(\checkmark(\vec{X}, \rho)))) \right\}.$$

Theorem 3.I.11. *Consider a* preg$^+$ *system* $G = (\Sigma, L, \mathcal{R})$ *such that* $FTP \sqsubseteq G$. E_G *is sound and ground-complete for strong bisimilarity on* $T(\Sigma)$.

Proof. Soundness follows immediately – by construction, the left hand side of each instance of an equation of the form given in Definition 3.I.10 can do (perform a certain action / satisfy a certain predicate) everything the right hand side can and vice versa. Concerning completeness, obviously the right hand side of the equation is in h.n.f. □

3.I.5 Consistency requirements

Allowing a language designer the freedom to work both with existential and universal predicates has its cost. It is easy to end up with overspecified or underspecified systems. We provide, therefore, by means of examples, a list of consistency requirements that a *preg*$^+$ system needs to have. We do not claim that this list is complete.

Example 3.I.12. *If the system has the rule* $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$*, it also needs to include the rule* $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a}}$.

Consistency Requirement 3.I.13. *Consider an operation* f *and a predicate* $P \in \mathcal{P}^{+\square}_{\exists}$. *The following must hold:*

$$\forall a \in \mathcal{A}^{\mathcal{D}}_{P} \;\; \textit{if} \;\; \frac{H}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]} \in \mathcal{R}^{\mathcal{A}}_{f} \;\; \textit{then} \;\; \frac{H}{P(f(\vec{x}))} \in \mathcal{R}^{\mathcal{P}}_{f}.$$

Example 3.I.14. *If the system includes the rules*

$$\frac{x \xrightarrow{a} x' \;\; y\not\rightarrow}{f(x,y) \xrightarrow{a} f(x',y)} \qquad \frac{x\not\rightarrow \;\; y \xrightarrow{a} y'}{f(x,y) \xrightarrow{a} f(x,y')}$$

and the predicate $\overset{a}{\dashrightarrow}$*, it also needs to include the rules:*

$$\frac{x \overset{a}{\dashrightarrow} \;\; y \overset{a}{\dashrightarrow}}{f(x,y) \overset{a}{\dashrightarrow}} \qquad \frac{x \overset{a}{\dashrightarrow} \;\; x\not\dashrightarrow}{f(x,y) \overset{a}{\dashrightarrow}} \qquad \frac{y \overset{a}{\dashrightarrow} \;\; y\not\dashrightarrow}{f(x,y) \overset{a}{\dashrightarrow}} \qquad \frac{x\not\dashrightarrow \;\; y\not\dashrightarrow}{f(x,y) \overset{a}{\dashrightarrow}}.$$

Example 3.I.15. *Consider* $\mathcal{A} = \{a, b, c\}$. *If the system includes the rules*

$$\forall \alpha \in \mathcal{A}: \quad \frac{x \xrightarrow{\alpha} x'}{x \parallel y \xrightarrow{\alpha} x' \parallel y} \qquad \frac{y \xrightarrow{\alpha} y'}{x \parallel y \xrightarrow{\alpha} x \parallel y'}$$

and the predicate $\Uparrow_a$ *(can immediately only perform a), it also needs to include the rule*

$$\frac{x \overset{b}{\dashrightarrow} \;\; x \overset{c}{\dashrightarrow} \;\; y \overset{b}{\dashrightarrow} \;\; y \overset{c}{\dashrightarrow}}{(x \parallel y) \Uparrow_a}$$

and vice versa.

Consistency Requirement 3.I.16. *Consider an operation* f *and a predicate* $P \in \mathcal{P}^{+\square}_{\forall}$. *For any a in* $\mathcal{A} \setminus \mathcal{A}^{\mathcal{D}}_{P}$ *it must hold that if none of the rules in* $\mathcal{R}^{\mathcal{A}}_{f}$ *of the form* $\dfrac{H}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]}$

can be applied then system has enough rules to infer $P(f(\vec{x}))$ for any closed instantiation of $\vec{x}$.

Example 3.I.17. *If a system includes the rule* $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \in \mathcal{R}^{\mathcal{A}}_{\_\parallel\_}$ *and the predicate* $\not\downarrow_a$, *then it also needs to include the rule* $\dfrac{x \xrightarrow{a} x'}{(x' \parallel y)\not\downarrow_a}$.

Consistency Requirement 3.I.18. *Let* $P \in \mathcal{P}^{\perp\!\!\!\sqcup}$.

$$\forall a \in \mathcal{A}^{\mathcal{D}}_{P} \;\; \textit{if} \;\; \frac{H}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]} \in \mathcal{R}^{\mathcal{A}}_{f} \;\; \textit{then} \;\; \frac{H}{P(f(\vec{x}))} \in \mathcal{R}^{\mathcal{P}}_{f}.$$

3.I.6 Concluding remarks

It is fair to say that the analysis in the current appendix indicates that an abstract, general treatment of arbitrary predicates is hard in the setting of the meta-theory of SOS if one wants to generate axiomatizations automatically. This seems to indicate that an approach based on coding predicates as transition labels is easier to handle when dealing with arbitrary predicates.

Chapter 4

Algebraic Meta-Theory of Processes with Data

4.1 Introduction

Algebraic properties capture some key features of programming and specification constructs and can be used both as design principles (for the semantics of such constructs) as well as for verification of programs and specifications built using them. When given the semantics of a language, inferring properties such as commutativity, associativity and unit element, as well as deriving sets of axioms for reasoning on the behavioural equivalence of two processes constitute one of the cornerstones of process algebras [25, 127] and play essential roles in several disciplines for behavioural modeling and analysis such as term rewriting [24] and model checking [36].

For formalisms with a Structural Operational Semantics (SOS), there exists a rich literature on meta-theorems guaranteeing key algebraic properties (commutativity [117], associativity [61], zero and unit elements [10], idempotence [5], and distributivity [9]) by means of restrictions on the syntactic shape of the transition rules. At the same time, for GSOS, a restricted yet expressive form of SOS specifications, one can obtain a sound and ground-complete axiomatization modulo strong bisimilarity [6]. Supporting some form of data (memory or store) is a missing aspect of these existing meta-theorems, which bars applicability to the semantics of numerous programming languages and formalisms that do feature these aspects in different forms.

Contribution In this chapter we provide a natural and generic link between the meta-theory of algebraic properties and axiomatizations, and SOS with data for which we consider that one data state models the whole memory. Namely, we move the data terms in SOS with data to the labels and instantiate them to closed terms; we call this process *currying*. Currying allows us to apply directly the existing rule formats for algebraic properties on the curried SOS specifications (which have process terms as states and triples of the form (datum, label, datum) as labels). We also present a new way of automatically deriving sound and ground-complete axiomatization schemas modulo strong bisimilarity for the curried systems for the setting in which the data component is characterized by constants. It turns out that strong bisimilarity for the curried SOS specification coincides with the notion of stateless bisimilarity in the original SOS specifications with data. The latter notion is extensively studied in [116] and used, among others, in [42, 75, 38, 39]. (This notion, in fact, coincides with the notion of strong bisimilarity proposed for Modular SOS in [110, Section 4.1].) Hence, using the existing rule formats, we can obtain algebraic laws for SOS specification with data that are sound with respect to stateless bisimilarity, as well as the weaker notions of initially stateless bisimilarity and statebased bisimilarity, studied in [116].

SOS with data and store has been extensively used in specifying semantics of programming and specification languages, dating back to the original work of Plotkin [123, 124]. Since then, several pieces of work have been dedicated to providing a formalization for SOS specification frameworks allowing one to include data and store and reason over it. The current chapter builds upon the approach proposed in [116] (originally published as [112]).

The idea of moving data from the configurations (states) of operational semantics to labels is reminiscent of Modular SOS [110, 109], Enhanced SOS [65], the Tile Model [74], and context-dependent-behaviour framework of [60]. The idea has also been applied in instances of SOS specification, such as those reported in [34, 38, 119]. The present chapter contributes to this body of knowledge by presenting a generic transformation from SOS specifications with data and store (as part of the configuration) to Transition System Specifications [86, 49]. The main purpose of this generic transformation is to enable exploiting the several existing rule formats defined on transition system specifications on the results of the transformation and then, transform the results back to the original SOS specifications (with data and store in the configuration) using a meaningful and well-studied notion of bisimilarity with data. Our transformation is also inspired

by the translation of SOS specifications of programming languages into rewriting logic, see e.g., [101, 100].

Structure The rest of this chapter is organized as follows. In Section 4.2, we recall some basic definitions regarding SOS specifications and behavioural equivalences. In Section 4.3, we present the currying technique and formulate the theorem regarding the correspondence between strong and stateless bisimilarity. In Section 4.4 we show how to obtain sound and ground-complete axiomatizations modulo strong bisimilarity for those curried systems for which the domain of the data component is a finite set of constants. We apply the currying technique to Linda [56], a coordination language from the literature chosen as case study in Section 4.5, and show how key algebraic properties of the operators defined in the language semantics are derived. We conclude the chapter in Section 4.6, by summarizing the results and presenting some directions for future work.

4.2 Preliminaries

4.2.1 Transition Systems Specifications

We assume a multisorted signature Σ with designated and distinct sorts P and D for processes and data, respectively. Moreover, we assume infinite and disjoint sets of process variables V_P (typical members: $x_P, y_P, x_{P_i}, y_{P_i} \ldots$) and data variables V_D (typical members: $x_D, y_D, x_{D_i}, y_{D_i} \ldots$), ranging over their respective sorts P and D.

Process and data signatures, denoted respectively by $\Sigma_P \subseteq \Sigma$ and $\Sigma_D \subseteq \Sigma$, are sets of function symbols with fixed arities. We assume in the remainder that the function symbols in Σ_D take only parameters of the sort Σ_D, while those in Σ_P can take parameters both from Σ_P and Σ_D, as in practical specifications of systems with data, process function symbols do take data terms as their parameters.

Terms are built using variables and function symbols by respecting their domains of definition. The sets of open process and data terms are denoted by $\mathbb{T}(\Sigma_P)$ and $\mathbb{T}(\Sigma_D)$, respectively. Disjointness of process and data variables is mostly for notational convenience. Function symbols from the process signature are typically denoted by f_P, g_P, f_{P_i} and g_{P_i}. Process terms are typically denoted by t_P, t'_P, and t_{P_i}. Function symbols from the data signature are typically denoted by f_D, f'_D and f_{D_i},

and data terms are typically denoted by t_D, t'_D, and t_{D_i}. The sets of closed process and data terms are denoted by $T(\Sigma_P)$ and $T(\Sigma_D)$, respectively. Closed process and data terms are typically denoted by p, q, p', p_i, p'_i and d, e, d', d_i, d'_i, respectively. We denote process and data substitutions by σ, σ', and ξ, ξ', respectively. We call substitutions $\sigma : V_P \to \mathbb{T}(\Sigma_P)$ process substitutions and $\xi : V_D \to \mathbb{T}(\Sigma_D)$ data substitutions. A substitution replaces a variable in an open term with another (possibly open) term. Notions of open and closed and the concept of substitution are lifted to formulae in the natural way.

Definition 4.2.1 (Transition System Specification)**.** *Consider a signature Σ and a set of labels L (with typical members $l, l', l_0, \ldots$). A* positive transition formula *is a triple (t, l, t'), where $t, t' \in \mathbb{T}(\Sigma)$ and $l \in L$, written $t \xrightarrow{l} t'$, with the intended meaning: process t performs the action labeled as l and becomes process t'.*

A transition rule *is defined as a tuple (H, α), where H is a set of formulae and α is a formula. The formulae from H are called* premises *and the formula α is called the* conclusion*. A transition rule is mostly denoted by $\dfrac{H}{\alpha}$ and has the following generic shape:*

$$\textbf{(d)}\ \frac{\{t_i \xrightarrow{l_{ij}} t_{ij} \mid i \in I, j \in J_i\}}{t \xrightarrow{l} t'},$$

where I, J_i are sets of indexes, $t, t', t_i, t_{ij} \in \mathbb{T}(\Sigma)$, and $l_{ij} \in L$. A transition system specification *(abbreviated TSS) is a tuple $(\Sigma, L, \mathcal{R})$ where Σ is a signature, L is a set of labels, and $\mathcal{R}$ is a set of transition rules of the provided shape.*

We extend the shape of a transition rule to handle process terms paired with data terms in the following manner:

$$\textbf{(d')}\ \frac{\{(t_{P_i}, t_{D_i}) \xrightarrow{l_{ij}} (t_{P_{ij}}, t_{D_{ij}}) \mid i \in I, j \in J_i\}}{(t_P, t_D) \xrightarrow{l} (t'_P, t'_D)},$$

where I, J_i are index sets, $t_P, t'_P, t_{P_i}, t_{P_{ij}} \in \mathbb{T}(\Sigma_P)$, $t_D, t'_D, t_{D_i}, t_{D_{ij}} \in \mathbb{T}(\Sigma_D)$, and $l_{ij} \in L$. A transition system specification with data *is a triple $\mathcal{T} = (\Sigma_P \cup \Sigma_D, L, \mathcal{R})$ where Σ_P and Σ_D are process and data signatures respectively, L is a set of labels, and $\mathcal{R}$ is a set of transition rules handling pairs of process and data terms.*

Definition 4.2.2. *Let $\mathcal{T}$ be a TSS with data. A* proof *of a formula ϕ from $\mathcal{T}$ is an upwardly branching tree whose nodes are labelled by formulas such that*

1. *the root node is labelled by ϕ, and*

2. *if ψ is the label of a node q and the set $\{\psi_i \mid i \in I\}$ is the set of labels of the nodes directly above q, then there exist a deduction rule $\frac{\{\chi_i \mid i \in I\}}{\chi}$, a process substitution σ, and a data substitution ξ such that the application of these substitutions to χ gives the formula ψ, and for all $i \in I$, the application of the substitutions to χ_i gives the formula ψ_i.*

Note that by removing the data substitution ξ from above we obtain the definition for proof of a formula from a standard TSS. The notation $\mathcal{T} \vdash \phi$ expresses that there exists a proof of the formula ϕ from the TSS (with data) $\mathcal{T}$. Whenever $\mathcal{T}$ is known from the context, we will write ϕ directly instead of $\mathcal{T} \vdash \phi$.

4.2.2 Bisimilarity

In this chapter we use two notions of equivalence over processes, one for standard transition system specifications and one for transition system specifications with data. Stateless bisimilarity is the natural counterpart of strong bisimilarity, used in different formalisms such as [38, 39, 42, 75].

Definition 4.2.3 (Strong Bisimilarity [120])**.** *Consider a TSS $\mathcal{T} = (\Sigma_P, L, \mathcal{R})$. A relation $R \subseteq T(\Sigma_P) \times T(\Sigma_P)$ is a* strong bisimulation *if and only if it is symmetric and $\forall_{p,q}\ (p,q) \in R \Rightarrow (\forall_{l,p'}\ p \xrightarrow{l} p' \Rightarrow \exists_{q'}\ q \xrightarrow{l} q' \wedge (q,q') \in R)$. Two closed terms p and q are* strongly bisimilar*, denoted by $p \leftrightarrows^{\mathcal{T}} q$ if there exists a strong bisimulation relation R such that $(p,q) \in R$.*

Definition 4.2.4 (Stateless Bisimilarity [116])**.** *Consider a TSS with data $\mathcal{T} = (\Sigma_P \cup \Sigma_D, L, \mathcal{R})$. A relation $R_{\mathrm{sl}} \subseteq T(\Sigma_P) \times T(\Sigma_P)$ is a* stateless bisimulation *if and only if it is symmetric and $\forall_{p,q}(p,q) \in R_{\mathrm{sl}} \Rightarrow \forall_{d,l,p',d'}(p,d) \xrightarrow{l} (p',d') \Rightarrow \exists_{q'}(q,d) \xrightarrow{l} (q',d') \wedge (p',q') \in R_{\mathrm{sl}}$. Two closed process terms p and q are* stateless bisimilar*, denoted by $p \leftrightarrows_{sl}^{\mathcal{T}} q$, if there exists a stateless bisimulation relation R_{sl} such that $(p,q) \in R_{\mathrm{sl}}$.*

4.2.3 Rule Formats for Algebraic Properties

As already stated, the literature on rule formats guaranteeing algebraic properties is extensive. For the purpose of this chapter we show the detailed line of reasoning only for the commutativity of binary operators, while, for readability, we refer to the corresponding papers and theorems for the other results in Section 4.5. We only present the format for binary operations. (See Section 5.3 for a more general one.)

Definition 4.2.5 (Commutativity)**.** *Given a TSS and a binary process operator f in its process signature, f is called* commutative w.r.t. $\sim$*, if the following equation is sound w.r.t. $\sim$:*

$$f(x_0, x_1) = f(x_1, x_0).$$

Definition 4.2.6 (Commutativity format [23])**.** *A transition system specification over signature Σ is in* comm-form *format with respect to a set of binary function symbols $COMM \subseteq \Sigma$ if all its f-defining transition rules with $f \in COMM$ have the following form*

$$\textbf{(c)}\ \frac{\{x_j \xrightarrow{l_{ij}} y_{ij} \mid i \in I\}}{f(x_0, x_1) \xrightarrow{l} t}$$

where $j \in \{0, 1\}$, I is an arbitrary index set, and variables appearing in the source of the conclusion and target of the premises are all pairwise distinct. We denote the set of premises of **(c)** *by H and the conclusion by α. Moreover, for each such rule, there exist a transition rule* **(c′)** *of the following form in the transition system specification*

$$\textbf{(c′)}\ \frac{H'}{f(x'_0, x'_1) \xrightarrow{l} t'}$$

and a bijective mapping (substitution) $\hbar$ on variables such that

- $\hbar(x'_0) = x_1$ *and* $\hbar(x'_1) = x_0$,
- $\hbar(t') \sim_{cc} t$ *and*
- $\hbar(h') \in H$, *for each* $h' \in H'$,

where $\sim_{cc}$ means equality up to swapping of arguments of operators in COMM in any context. Transition rule **(c′)** *is called the* commutative mirror *of* **(c)**.

Theorem 4.2.1 (Commutativity for comm-form [23])**.** *If a transition system specification is in* comm-form *format with respect to a set of operators COMM, then all operators in COMM are commutative with respect to strong bisimilarity.*

4.2.4 Sound and ground-complete axiomatizations

In this section we recall several key aspects presented in [6], where the authors provide a procedure for converting any GSOS language definition that disjointly extends the language for synchronization trees to a finite complete equational axiom system which characterizes strong bisimilarity over a disjoint extension of

the original language. It is important to note that we work with the GSOS format because it guarantees that bisimilarity is a congruence and that the transition relation is finitely branching [49]. For the sake of simplicity, we confine ourselves to the positive subset of the GSOS format; we expect the generalization to the full GSOS format to be straightforward.

Definition 4.2.7 (Positive GSOS rule format)**.** *Consider a process signature Σ_P. A* positive GSOS rule *ρ over Σ_P has the shape:*

$$\textbf{(g)}\ \frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, j \in J_i\}}{f(x_1, \ldots, x_n) \xrightarrow{l} C[\vec{x}, \vec{y}]},$$

where all variables are distinct, f is an operation symbol form Σ_P with arity n, $I \subseteq \{1, \ldots, n\}$, J_i finite for each $i \in I$, l_{ij} and l are labels standing for actions ranging over a given set denoted by L, and $C[\vec{x}, \vec{y}]$ is a Σ_P-context with variables including at most the x_i's and y_{ij}'s.

A finite tree term t is built according to the following grammar:

$$t ::= \mathbf{0} \mid l.t\ (\forall l \in L) \mid t + t.$$

We denote this signature by Σ_{BCCSP}. Intuitively, $\mathbf{0}$ represents a process that does not exhibit any behaviour, $s + t$ is the nondeterministic choice between the behaviours of s and t, while $l.t$ is a process that first performs action l and behaves like t afterwards. The operational semantics that captures this intuition is given by the rules of BCCSP [79]:

$$\frac{}{l.x \xrightarrow{l} x} \qquad \frac{x \xrightarrow{l} x'}{x + y \xrightarrow{l} x'} \qquad \frac{y \xrightarrow{l} y'}{x + y \xrightarrow{l} y'}.$$

Definition 4.2.8 (Axiom System)**.** *An* axiom *(or* equation*)* system *E over a signature Σ is a set of equalities of the form $t = t'$, where $t, t' \in \mathbb{T}(\Sigma)$. An equality $t = t'$, for some $t, t' \in \mathbb{T}(\Sigma)$, is derivable from E, denoted by $E \vdash t = t'$, if and only if it is in the smallest congruence relation over Σ-terms induced by the equalities in E.*

We consider the axiom system E_{BCCSP} which consists of the following axioms:

$$x + y = y + x \qquad x + x = x$$
$$(x + y) + z = x + (y + z) \qquad x + \mathbf{0} = x\,.$$

Theorem 4.2.2. *E_{BCCSP} is sound and ground-complete for bisimilarity on $T(\Sigma_{\mathrm{BCCSP}})$. That is, it holds that $E_{\mathrm{BCCSP}} \vdash p = q$ if, and only if, $p \leftrightarrows^{\mathrm{BCCSP}} q$ for any two ground terms p and $q \in T(\Sigma_{\mathrm{BCCSP}})$.*

Definition 4.2.9 (Disjoint extension)**.** *A GSOS system G' is a disjoint extension of a GSOS system G, written $G \sqsubseteq G'$, if the signature and the rules of G' include those of G, and G' does not introduce new rules for operations in G.*

In [6] it is elaborated how to obtain an axiomatization for a GSOS system G that disjointly extends BCCSP. For technical reasons the procedure involves initially transforming G into a new system G' that conforms to a restricted version of the GSOS format, named *smooth and distinctive*. We avoid presenting this restricted format, as the method proposed in Section 4.4 allows us to obtain the axiomatization without the need to transform the initial system G.

4.3 Currying Data

We apply the process of currying [133] known from functional programming to factor out the data from the source and target of transitions and enrich the label to a triple capturing the data flow of the transition. This shows that for specifying behaviour and data of dynamic systems, the data may be freely distributed over states (as part of the process terms) or system dynamics (action labels of the transition system), providing a natural correspondence between the notions of stateless bisimilarity and strong bisimilarity. An essential aspect of our approach is that the process of currying is a syntactic transformation defined on transition system specifications (and not a semantic transformation on transition systems); this allows us to apply meta-theorems from the meta-theory of SOS and obtain semantic results by considering the syntactic shape of (transformed) SOS rules.

Definition 4.3.1 (Currying and Label Closure)**.** *Consider the TSS with data $\mathcal{T} = (\Sigma_P \cup \Sigma_D, L, \mathcal{R})$ and transition rule $\rho \in \mathcal{R}$ of the shape $\rho = \dfrac{\{(t_{P_i}, t_{D_i}) \xrightarrow{l_{ij}} (t_{P_{ij}}, t_{D_{ij}}) \mid i \in I, j \in J_i\}}{(t_P, t_D) \xrightarrow{l} (t'_P, t'_D)}$.*

The curried version *of ρ is the rule $\rho^c = \dfrac{\{t_{P_i} \overset{(t_{D_i}, l_{ij}, t_{D_{ij}})}{\longrightarrow} t_{P_{ij}} \mid i \in I, j \in J_i\}}{t_P \overset{(t_D, l, t'_D)}{\longrightarrow} t'_P}$. We further define $\mathcal{R}^c = \{\rho^c \mid \rho \in \mathcal{R}\}$ and $L^c = \{(t_D, l, t'_D) \mid l \in L,\ \ t_D, t'_D \in \mathbb{T}(\Sigma_D)\}$. The* curried version *of $\mathcal{T}$ is defined as $\mathcal{T}^c = (\Sigma_P, L^c, \mathcal{R}^c)$.*

By $\rho^c_\xi = \dfrac{\{t_{P_i} \overset{(\xi(t_{D_i}),l_{ij},\xi(t_{D_{ij}}))}{\longrightarrow} t_{P_{ij}} \mid i \in I, j \in J_i\}}{t_P \overset{(\xi(t_D),l,\xi(t'_D))}{\longrightarrow} t'_P}$ *we denote the* closed label version *of* ρ^c *with respect to the closed data substitution* ξ*. By* $cl(\rho^c)$ *we denote the set consisting of all closed label versions of* ρ^c*, i.e.* $cl(\rho^c) = \{\rho^c_\xi \mid \rho^c \in \mathcal{R}^c, \xi \text{ is a closed data substitution}\}$*. We further define* $cl(\mathcal{R}^c) = \{cl(\rho^c) \mid \rho^c \in \mathcal{R}^c\}$ *and* $cl(L^c) = \{(\xi(t_D), l, \xi(t'_D)) \mid (t_D, l, t'_D) \in L^c, \xi \text{ is a closed data substitution}\}$*. The* closed label version *of* $\mathcal{T}^c$ *is denoted by* $cl(\mathcal{T}^c) = (\Sigma_P, cl(L^c), cl(\mathcal{R}^c))$.

Our goal is to reduce the notion of stateless bisimilarity between two closed process with data terms to strong bisimilarity by means of currying the TSS with data and closing its labels. The following theorem states how this goal can be achieved.

Theorem 4.3.1. *Given a TSS* $\mathcal{T} = (\Sigma, L, D)$ *with data, for each two closed process terms* $p, q \in T(\Sigma_P)$*,* $p \leftrightarrows^{\mathcal{T}}_{sl} q$ *if, and only if,* $p \leftrightarrows^{cl(\mathcal{T}^c)} q$.

We provide a proof for the theorem in Appendix 4.A.

4.4 Axiomatizing GSOS with Data

In this section, we provide an axiomatization schema for reasoning about stateless bisimilarity. We find it easier to work directly with curried systems instead of systems with data because this allows us to adapt the method introduced in [6] by considering the set of more complex labels that integrate the data, as presented in Section 4.3.

It is important to note that we present the schema by considering that the signature for data terms, Σ_D, consists only of a finite set of constants.

BCCSP is extended to a setting with data, BCCSP_D. This is done by adding to the signature for process terms Σ_{BCCSP} two auxiliary operators for handling the store, named *check* and *update*, obtaining a new signature, Σ_{BCCSP_D}. Terms over Σ_{BCCSP_D} are built according to the following grammar:

$$t_P ::= \mathbf{0} \mid l.t_P \quad \forall_{l \in L} \mid check(d, t_P) \mid update(d, t_P) \mid t_P + t_P.$$

Intuitively, operation $check(d, t_P)$ makes sure that, before executing an initial action from t_P, the store has the value d, and $update(d, t_P)$ changes the store value to d after executing an initial action of process t_P. The prefix operation does not affect the store. We directly provide the curried set of rules defining the semantics of

BCCSP^c_D.

$$\frac{}{l.x_P \xrightarrow{(x_D,l,x_D)} x_P} \qquad \frac{x_P \xrightarrow{(x_D,l,x'_D)} x'_P}{check(x_D,x_P) \xrightarrow{(x_D,l,x'_D)} x'_P} \qquad \frac{x_P \xrightarrow{(x_D,l,x'_D)} x'_P}{update(y_D,x_P) \xrightarrow{(x_D,l,y_D)} x'_P}$$

$$\frac{x_P \xrightarrow{(x_D,l,x'_D)} x'_P}{x_P + y_P \xrightarrow{(x_D,l,x'_D)} x'_P} \qquad \frac{y_P \xrightarrow{(x_D,l,x'_D)} y'_P}{x_P + y_P \xrightarrow{(x_D,l,x'_D)} y'_P}.$$

Definition 4.2.3 can easily be adapted to the setting of SOS systems with data.

Definition 4.4.1. *Consider a TSS* $\mathcal{T} = (\Sigma_P \cup \Sigma_D, L, \mathcal{R})$*, which means that* $\mathcal{T}^c = (\Sigma_P, L^c, \mathcal{R}^c)$*. A relation* $R \subseteq T(\Sigma_P) \times T(\Sigma_P)$ *is a* strong bisimulation *if and only if it is symmetric and* $\forall_{p,q}\ (p,q) \in R \Rightarrow \forall_{d,l,d',p'}\ p \overset{(d,l,d')}{\rightarrow} p' \Rightarrow \exists_{q'}\ q \overset{(d,l,d')}{\rightarrow} q' \wedge (q,q') \in R$*. Two closed terms* p *and* q *are* strongly bisimilar*, denoted by* $p \leftrightarrows^{\mathcal{T}^c} q$ *if there exists a strong bisimulation relation* R *such that* $(p,q) \in R$*.*

The axiomatization $E_{\mathrm{BCCSP}^c_D}$ of strong bisimilarity over BCCSP^c_D, which is to be proven sound and ground-complete in the remainder of this section, is given below:

$x_P + y_P$	$=$	$y_P + x_P$	(n-comm)
$x_P + (y_P + z_P)$	$=$	$(x_P + y_P) + z_P$	(n-assoc)
$x_P + x_P$	$=$	x_P	(n-idem)
$x_P + \mathbf{0}$	$=$	x_P	(n-zero)
$check(x_D, x_P + y_P)$	$=$	$check(x_D,x_P) + check(x_D,y_P)$	(nc)
$update(x_D, x_P + y_P)$	$=$	$update(x_D,x_P) + update(x_D,y_P)$	(nu)
$check(x_D, update(y_D,x_P))$	$=$	$update(y_D, check(x_D,x_P))$	(cu)
$update(x_D, update(y_D,x_P))$	$=$	$update(x_D,x_P)$	(uu)
$check(d, check(d,x_P))$	$=$	$x_P \quad (\forall d \in \Sigma_D)$	(cc)
$check(d, check(d',x_P))$	$=$	$\mathbf{0} \quad (\forall d,d' \in \Sigma_D, d \neq d')$	(cc')
$l.x_P$	$=$	$\sum_{d\in\Sigma_D} update(d, check(d, l.x_P))$	(lc)

Recall that Σ_D is a finite set of constants, and, therefore, the right hand side of axiom (lc) has a finite number of summands.

The following theorem is proved in the standard fashion.

Theorem 4.4.1 (Soundness)**.** *For each two terms* s,t *in* $\mathbb{T}(\Sigma_{\mathrm{BCCSP}^c_D})$ *it holds that if* $E_{\mathrm{BCCSP}^c_D} \vdash s = t$ *then* $s \leftrightarrows^{\mathrm{BCCSP}^c_D} t$*.*

We now introduce the concept of terms in *head normal form*, which is essential for proving the completeness of axiom systems.

Definition 4.4.2 (Head Normal Form)**.** *Let Σ_P be a signature such that $\Sigma_{\mathrm{BCCSP}^c_D} \subseteq \Sigma_P$. A term t in $\mathbb{T}(\Sigma_P)$ is in* head normal form *(for short, h.n.f.) if*

$$t = \sum_{i \in I} update(t'_{Di}, check(t_{Di}, l_i.t_{Pi})),$$

where, for every $i \in I$, $t_{Di}, t'_{Di} \in \mathbb{T}(\Sigma_D)$, $t_{Pi} \in \mathbb{T}(\Sigma_D)$, $l_i \in L$. The empty sum ($I = \emptyset$) is denoted by the deadlock constant $\mathbf{0}$.

Lemma 4.4.3 (Head Normalization)**.** *For any term p in $T(\Sigma_{\mathrm{BCCSP}^c_D})$, there exists p' in $T(\Sigma_{\mathrm{BCCSP}^c_D})$ in h.n.f. such that $E_{\mathrm{BCCSP}^c_D} \vdash p = p'$.*

Proof. By induction on the number of symbols appearing in p. We proceed with a case distinction on the head symbol of p.

Base case

- $p = \mathbf{0}$; this case is vacuous, because p is already in h.n.f.

Inductive step cases

- p is of the shape $l.p'$; then

  $p \overset{\text{def. } p}{=} l.p' \overset{(\mathrm{lc})}{=} \sum_{d \in T(\Sigma_D)} update(d, check(d, l.p'))$, which is in h.n.f.

- p is of the shape $check(d'', p'')$; then

  $p \overset{\text{def. } p}{=} check(d'', p'') \overset{\text{ind. hyp.}}{=}$
  $check(d'', \sum_{i \in I} update(d'_i, check(d_i, l_i.p'_i))) \overset{(\mathrm{nc})}{=}$
  $\sum_{i \in I} check(d'', update(d'_i, check(d_i, l_i.p'_i))) \overset{(\mathrm{cu})}{=}$
  $\sum_{i \in I} update(d'_i, check(d'', check(d_i, l_i.p'_i))) \overset{(\mathrm{cc,cc'})}{=}$
  $\sum_{i \in I, d_i = d''} update(d'_i, check(d_i, l_i.p'_i))$, which is in h.n.f.

- p is of the form $update(d'', p'')$; then

  $p \overset{\text{def. } p}{=} update(d'', p'') \overset{\text{ind. hyp.}}{=}$
  $update(d'', \sum_{i \in I} update(d'_i, check(d_i, l_i.p'_i))) \overset{(\mathrm{nu})}{=}$
  $\sum_{i \in I} update(d'', update(d'_i, check(d_i, l_i.p'_i))) \overset{(\mathrm{uu})}{=}$
  $\sum_{i \in I} update(d'', check(d_i, l_i.p'_i))$, which is in h.n.f.

- p is of the form $p_0 + p_1$; then

$p \overset{\text{def. } p}{=} p_0 + p_1 \overset{\text{ind. hyp.}}{=}$
$\sum_{i\in I} check(d'', update(d'_i, check(d_i, l_i.p'_i))) +$
$\sum_{j\in J} check(d'', update(d'_j, check(d_j, l_j.p'_j))) =$
$\sum_{k\in I\cup J} check(d'', update(d'_k, check(d_k, l_k.p'_k)))$, which is in h.n.f.

□

Theorem 4.4.2 (Ground-completeness)**.** *For each two closed terms $p, q \in T(\Sigma_{\text{BCCSP}^c_D})$, it holds that if $p \leftrightarrows^{\text{BCCSP}^c_D} q$, then $E_{\text{BCCSP}^c_D} \vdash p = q$.*

Proof. We assume, by Lemma 4.4.3 that p, q are in h.n.f., define the function *height* as follows:

$$height(p) = \begin{cases} 0 & \text{if } p = \mathbf{0} \\ 1 + \max(height(p_1), height(p_2)) & \text{if } p = p_1 + p_2 \\ 1 + height(p') & \text{if } p = update(d', check(d, l.p')), \end{cases}$$

and prove the property by induction on $M = \max(height(p), height(q))$.

Base case ($M = 0$) This case is vacuous, because $p = q = \mathbf{0}$, so $E_{\text{BCCSP}^c_D} \vdash p = q$.

Inductive step case ($M > 0$) We prove $E_{\text{BCCSP}^c_D} \vdash p = q + p$ by arguing that every summand of q is provably equal to a summand of p. Let $update(d', check(d, l.q'))$ be a summand of q. By applying the rules defining BCCSP^c_D, we derive $q \overset{(d,l,d')}{\to} q'$. As $q \leftrightarrows^{\text{BCCSP}^c_D} p$ holds, it has to be the case that $p \overset{(d,l,d')}{\to} p'$ and $q' \leftrightarrows^{\text{BCCSP}^c_D} p'$ hold. As $\max(height(q'), height(p')) < M$, from the inductive hypothesis it results that $E_{\text{BCCSP}^c_D} \vdash q' = p'$, hence $update(d', check(d, l.q'))$ is provably equal to the term $update(d', check(d, l.p'))$, which is a summand of p.

It follows, by symmetry, that $E_{\text{BCCSP}^c_D} \vdash q = p + q$ holds, which ultimately leads to the fact that $E_{\text{BCCSP}^c_D} \vdash p = q$ holds. □

Consider a TSS with data $\mathcal{T} = (\Sigma_P \cup \Sigma_D, L, \mathcal{R})$. For an operation $f \in \Sigma_P$, we denote by $\mathcal{R}_f$ the set of all rules defining f. All the rules in $\mathcal{R}_f$ are in the GSOS format extended with the data component. For the simplicity of presenting the axiomatization schema, we assume that f only has process terms as arguments, baring in mind that adding data terms is trivial.

When given a signature Σ_P that includes $\Sigma_{\mathrm{BCCSP}^c_D}$, the purpose of an axiomatization for a term $p \in T(\Sigma_P)$ is to derive another term p' such that $p \leftrightarrows^{\mathcal{T}^c} p'$ and $p' \in T(\Sigma_{\mathrm{BCCSP}^c_D})$.

Definition 4.4.4 (Axiomatization schema)**.** *Consider a TSS $\mathcal{T}^c = (\Sigma_P, L^c, \mathcal{R}^c)$ such that $\mathrm{BCCSP}^c_D \sqsubseteq \mathcal{T}^c$. By $E_{\mathcal{T}^c}$ we denote the axiom system that extends $E_{\mathrm{BCCSP}^c_D}$ with the following axiom schema for every operation f in $\mathcal{T}$, parameterized over the vector of closed process terms $\vec{p}$ in h.n.f.:*

$$f(\vec{p}) = \sum \left\{ update(d', check(d, l.C[\vec{p}, \vec{y_P}])) \;\middle|\; \rho = \frac{H}{f(\vec{p}) \xrightarrow{(d,l,d')} C[\vec{p}, \vec{q}]} \in cl(\mathcal{R}^c_f) \text{ and } \checkmark(\vec{p}, \rho) \right\},$$

where $\checkmark$ is defined as $\checkmark(\vec{p}, \rho) = \bigwedge_{p_k \in \vec{p}} \checkmark'(p_k, k, \rho)$,

and $$\checkmark'\left(p_k, k, \frac{\{x_{P_i} \overset{(d_i, l_{ij}, d'_{ij})}{\rightarrow} y_{P_{ij}} \mid i \in I, j \in J_i\}}{f(\vec{p}) \xrightarrow{(d,l,d')} C[\vec{p}, \vec{q}]}\right) =$$

$$\text{if } k \in I \text{ then } \forall_{j \in J_k} \exists_{p', p''} \, E_{\mathrm{BCCSP}^c_D} \vdash p_k = update(d'_{kj}, check(d_k, l_{kj}.p')) + p''.$$

Intuitively, the axiom transforms $f(\vec{p})$ into a sum of closed terms covering all its execution possibilities. We iterate, in order to obtain them, through the set of f-defining rules and check if $\vec{p}$ satisfies their hypotheses by using the meta-operation $\checkmark$. $\checkmark$ makes sure that, for a given rule, every component of $\vec{p}$ is a term with enough action prefixed summands satisfying the hypotheses associated to that component. Note that the axiomatization is built in such a way that it always derives terms in head normal form. Also note that the sum on the right hand side is finite because of our initial assumption that the signature for data is a finite set of constants.

The reason why we conceived the axiomatization in this manner is of practical nature. Our past experience shows that this type of schemas may bring terms to their normal form faster than finite axiomatizations. Aside this, we do not need to transform the initial system, as presented in [6].

Theorem 4.4.3. *Consider a TSS $\mathcal{T}^c = (\Sigma_P, L^c, \mathcal{R}^c)$ such that $\mathrm{BCCSP}^c_D \sqsubseteq \mathcal{T}^c$. $E_{\mathcal{T}^c}$ is sound and ground-complete for strong bisimilarity on $T(\Sigma_P)$.*

Proof. It is easy to see that, because of the head normal form of the right hand side of every axiom, the completeness of the axiom schema reduces to the completeness proof for bisimilarity on $T(\Sigma_{\mathrm{BCCSP}^c_D})$.

In order to prove the soundness, we denote, for brevity, the right hand side of the schema in Definition 4.4.4 by *RHS*.

Let us first prove that if $f(\vec{p})$ performs a transition then it can be matched by *RHS*. Consider a rule $\rho \in cl(\mathcal{R}_f^c)$ that can be applied for $f(\vec{p})$: $\rho = \dfrac{\{x_i \overset{(d_i,l_{ij},d_{ij})}{\rightarrow} y_{ij} \mid i \in I, j \in J_i\}}{f(\vec{x}) \overset{(d,l,d')}{\rightarrow} C[\vec{x},\vec{y}]}$. Then $f(\vec{p}) \overset{(d,l_{ij},d')}{\rightarrow} C[\vec{p},\vec{q}]$ holds and, at the same time, all of the rule's premises are met. This means that p_i is of the form $\sum_{j\in J_i} update(d_{ij}, check(d_i, l_{ij}.p_{ij})) + p'$ for some p' and p_{ij}'s. It is easy to see that all the conditions for $\checkmark$ are met, so $(d,l,d').C[\vec{p},\vec{q}]$ is a summand of *RHS*, and therefore it holds that $RHS \overset{(d_i,l_{ij},d_{ij})}{\rightarrow} C[\vec{p},\vec{q}]$, witch matches the transition from $f(\vec{p})$.

The proof for the fact that $f(\vec{p})$ can match any of the transitions of *RHS* is similar. □

We end this section with the remark that the problem of extending the axiomatization schema to the setting with arbitrary data terms is still open. The most promising solution we have thought of involves using the infinite alternative quantification operation from [125]. This operation would help us to define and express head normal forms as (potentially) infinite sums, parameterized over data variables [25].

4.5 Case Study: The Coordination Language Linda

In what follows we present the semantics and properties of a core prototypical language.

The provided specification defines a structural operational semantics for the coordination language Linda; the specification is taken from [116] and is a slight adaptation of the original semantics presented in [55] (by removing structural congruences and introducing a terminating process ε). Process constants (atomic process terms) in this language are ε (for terminating process), $ask(u)$ and $nask(u)$ (for checking existence and absence of tuple u in the shared data space, respectively), $tell(u)$ (for adding tuple u to the space) and $get(u)$ (for taking tuple u from the space). Process composition operators in this language include nondeterministic choice (+), sequential composition (;) and parallel composition (||). The data

signature of this language consists of a constant {} for the empty multiset and a class of unary function symbols $\cup\{u\}$, for all tuples u, denoting the union of a multiset with a singleton multiset containing tuple u. The operational state of a Linda program is denoted by (p, d) where p is a process term in the above syntax and d is a multiset modeling the shared data space.

The transition system specification defines one relation $\rightarrow$ and one predicate $\downarrow$. Note that $\rightarrow$ is unlabeled, unlike the other relations considered so far. Without making it explicit, we tacitly consider the termination predicate $\downarrow$ as a binary transition relation $\overset{\downarrow}{\rightarrow}$ with the pair (x_P, x_D), where x_P and x_D are fresh yet arbitrary process and data variables, respectively.

Below we provide a table consisting of both the original and the curried and closed label versions of the semantics of Linda on the left and, respectively, on the right.

$$(1)\ \frac{}{(\varepsilon, d)\downarrow} \qquad (1_c)\ \frac{}{\varepsilon\downarrow}$$

$$(2)\ \frac{}{(ask(u), d\cup\{u\})\rightarrow(\varepsilon, d\cup\{u\})} \qquad (2_c)\ \frac{}{ask(u)\overset{(d\cup\{u\},-,d\cup\{u\})}{\rightarrow}\varepsilon}$$

$$(3)\ \frac{}{(tell(u), d)\rightarrow(\varepsilon, d\cup\{u\})} \qquad (3_c)\ \frac{}{tell(u)\overset{(d,-,d\cup\{u\})}{\rightarrow}\varepsilon}$$

$$(4)\ \frac{}{(get(u), d\cup\{u\})\rightarrow(\varepsilon, d)} \qquad (4_c)\ \frac{}{get(u)\overset{(d\cup\{u\},-,d)}{\rightarrow}\varepsilon}$$

$$(5)\ \frac{}{(nask(u), d)\rightarrow(\varepsilon, d)}[u\notin d] \qquad (5_c)\ \frac{}{nask(u)\overset{(d,-,d)}{\rightarrow}\varepsilon}[u\notin d]$$

$$(6)\ \frac{(x_P, d)\downarrow}{(x_P + y_P, d)\downarrow} \quad (7)\ \frac{(y_P, d)\downarrow}{(x_P + y_P, d)\downarrow} \qquad (6_c)\ \frac{x_P\downarrow}{x_P + y_P\downarrow} \quad (7_c)\ \frac{y\downarrow}{x_P + y_P\downarrow}$$

$$(8)\ \frac{(x_P, d)\rightarrow(x'_P, d')}{(x_P + y_P, d)\rightarrow(x'_P, d')} \qquad (8_c)\ \frac{x_P\overset{(d,-,d')}{\rightarrow}x'_P}{x_P + y_P\overset{(d,-,d')}{\rightarrow}x'_P}$$

$$(9)\ \frac{(y_P, d)\rightarrow(y'_P, d')}{(x_P + y_P, d)\rightarrow(y'_P, d')} \qquad (9_c)\ \frac{y_P\overset{(d,-,d')}{\rightarrow}y'_P}{x_P + y_P\overset{(d,-,d')}{\rightarrow}y'_P}$$

$$(10)\ \frac{(x_P, d)\rightarrow(x'_P, d')}{(x_P\ ;\ y_P, d)\rightarrow(x'_P\ ;\ y_P, d')} \qquad (10_c)\ \frac{x_P\overset{(d,-,d')}{\rightarrow}x'_P}{x_P\ ;\ y_P\overset{(d,-,d')}{\rightarrow}x'_P\ ;\ y_P}$$

$$(11)\ \frac{(x_P, d)\downarrow\ \ (y_P, d)\rightarrow(y'_P, d')}{(x_P\ ;\ y_P, d)\rightarrow(y'_P, d')} \qquad (11_c)\ \frac{x_P\downarrow\quad y_P\overset{(d,-,d')}{\rightarrow}y'_P}{x_P\ ;\ y_P\overset{(d,-,d')}{\rightarrow}y'_P}$$

$$(12)\ \frac{(x_P, d)\downarrow\ (y_P, d)\downarrow}{(x_P\ ;\ y_P, d)\downarrow} \qquad (12_c)\ \frac{x_P\downarrow\ y_P\downarrow}{x_P\ ;\ y_P\downarrow}$$

$$(13)\ \frac{(x_P, d)\rightarrow(x'_P, d')}{(x_P \parallel y_P, d)\rightarrow(x' \parallel y, d')} \qquad (13_c)\ \frac{x_P \overset{(d,-,d')}{\rightarrow} x'_P}{x_P \parallel y_P \overset{(d,-,d')}{\rightarrow} x'_P \parallel y_P}$$

$$(14)\ \frac{(y_P, d)\rightarrow(y'_P, d')}{(x_P \parallel y_P, d)\rightarrow(x_P \parallel y'_P, d')} \qquad (14_c)\ \frac{y_P \overset{(d,-,d')}{\rightarrow} y'_P}{x_P \parallel y_P \overset{(d,-,d')}{\rightarrow} x_P \parallel y'_P}$$

$$(15)\ \frac{(x_P, d)\downarrow\ (y_P, d)\downarrow}{(x_P \parallel y_P, d)\downarrow} \qquad (15_c)\ \frac{x_P\downarrow\ y_P\downarrow}{x_P \parallel y_P\downarrow}$$

In the curried SOS rules, d and d' are arbitrary closed data terms, i.e., each transition rule given in the curried semantics represents a (possibly infinite) number of rules for each and every particular $d, d' \in T(\Sigma_D)$. It is worth noting that by using the I-MSOS framework [111] we can present the curried system without explicit labels at all as they are propagated implicitly between the premises and conclusion.

Consider transition rules (6_c), (7_c), (8_c), and (9_c); they are the only $+$-defining rules and they fit in the commutativity format of Definition 4.2.6. It follows from Theorem 4.2.1 that the equation $x + y = y + x$ is sound with respect to strong bisimilarity in the curried semantics. Subsequently, following Theorem 4.3.1, we have that the previously given equation is sound with respect to stateless bisimilarity in the original semantics. (Moreover, we have that $(x_0 + x_1, d) = (x_1 + x_0, d)$ is sound with respect to statebased bisimilarity for all $d \in T(\Sigma_D)$.)

Following a similar line of reasoning, we get that $x \parallel y = y \parallel x$ is sound with respect to stateless bisimilarity in the original semantics.

In addition, we derived the following axioms for the semantics of Linda, using the meta-theorems stated in the third column of the table. The semantics of sequential composition in Linda is identical to the sequential composition (without data) studied in Example 9 of [61]; there, it is shown that this semantics conforms to the Assoc-De Simone format introduced in [61] and hence, associativity of sequential composition follows immediately. Also semantics of nondeterministic choice falls within the scope of the Assoc-De Simone format (with the proposed coding of predicates), and hence, associativity of nondeterministic choice follows (note that in [61] nondeterministic choice without termination rules is treated in Example 1; moreover, termination rules in the semantics of parallel composition are discussed in Section 4.3 and shown to be safe for associativity). Following

a similar line of reasoning associativity of parallel composition follows from the conformance of its rules to the Assoc-De Simone format of [61]. Idempotence for + can be obtained, because rules (6_c), (7_c) and (8_c), (9_c) are choice rules [5, Definition 40] and the family of rules (6_c) to (9_c) for all data terms d and d' ensure that the curried specification is in idempotence format with respect to the binary operator +. The fact that ε is unit element for ; is proved similarly as in [10], Example 10.

Property	Axiom	Meta-Theorem
Associativity for ;	$x\ ;\ (y\ ;\ z) = (x\ ;\ y)\ ;\ z$	Theorem 1 of [61]
Associativity for +	$x + (y + z) = (x + y) + z$	Theorem 1 of [61]
Associativity for $\|\|$	$x \parallel (y \parallel z) = (x \parallel y) \parallel z$	Theorem 1 of [61]
Idempotence for +	$x + x = x$	Theorem 42 of [5]
Unit element for ;	$\varepsilon\ ;\ x = x$	Theorem 3 of [10]
Distributivity of + over ;	$(x + y)\ ;\ z = (x\ ;\ y) + (x\ ;\ z)$	Theorem 3 of [9]

We currently cannot derive an axiomatization for Linda because its semantics involves arbitrary data terms, as opposed to a finite number of constants.

4.6 Conclusions

In this chapter, we have proposed a generic technique for extending the meta-theory of algebraic properties to SOS with data, memory or store. In a nutshell, the presented technique allows for focusing on the structure of the process (program) part in SOS rules and ignoring the data terms in order to obtain algebraic properties, as well as, a sound and ground complete set of equations w.r.t. stateless bisimilarity. We have demonstrated the applicability of our method by means of the well known coordination language Linda.

It is also worth noting that one can check whether a system is in the process-tyft format presented in [115] in order to infer that stateless bisimilarity is a congruence, and if this is the case, then strong bisimilarity over the curried system is also a congruence. Our results are applicable to a large body of existing operators in the literature and make it possible to dispense with several lengthy and laborious soundness proofs in the future.

Our approach can be used to derive algebraic properties that are sound with respect to weaker notions of bisimilarity with data, such as initially stateless

and statebased bisimilarity [116]. We do expect to obtain stronger results, e.g., for zero element with respect to statebased bisimilarities, by scrutinizing data dependencies particular to these weaker notions. We would like to study coalgebraic definitions of the notions of bisimilarity with data (following the approach of [135]) and develop a framework for SOS with data using the bialgebraic approach. Furthermore, it is of interest to check how our technique can be applied to quantitative systems where non-functional aspects like probabilistic choice or stochastic timing is encapsulated as data. We also plan to investigate the possibility of automatically deriving axiom schemas for systems whose data component is given as arbitrary terms, instead of just constants.

4.A Proof of Theorem 4.3.1

Given a TSS $\mathcal{T} = (\Sigma, L, D)$ with data, for each two closed process terms $p, q \in T(\Sigma_P)$, $p \leftrightarrows^{\mathcal{T}}_{sl} q$ if, and only if, $p \leftrightarrows^{cl(\mathcal{T}^c)} q$.

Proof. Before we proceed with the proof of the theorem, we state and prove the following auxiliary lemma.

Lemma 4.A.1. *For each two closed process terms $p, p' \in T(\Sigma_P)$, each two closed data terms $d, d' \in T(\Sigma_D)$ and each label $l \in L$, it holds that $\mathcal{T} \vdash (p, d) \xrightarrow{l} (p', d')$ if and only if $cl(\mathcal{T}^c) \vdash p \xrightarrow{(d,l,d')} p'$.*

Proof. We split the bi-implication into two implications and prove them below:

- $\Rightarrow$ By induction on the depth of the proof for $\mathcal{T} \vdash (p, d) \xrightarrow{l} (p', d')$. Since the induction basis is a special case of the induction step (in which the last transition rule in the proof has no premises), we dispense with stating the induction basis separately. Assume that the last transition rule in the proof is
$$\rho = \frac{\{(t_{P_i}, t_{D_i}) \xrightarrow{l_{ij}} (t_{P_{ij}}, t_{D_{ij}}) \mid i \in I, j \in J_i\}}{(t_P, t_D) \xrightarrow{l} (t'_P, t'_D)}.$$
Then there exists a closed process substitution σ and a closed data substitution ξ such that $\sigma(t_P) = p$, $\sigma(t'_P) = p'$, $\xi(t_D) = d$, $\xi(t'_D) = d'$ and moreover $\mathcal{T} \vdash (\sigma(t_{P_i}), \xi(t_{D_i})) \xrightarrow{l_{ij}} (\sigma(t_{P_{ij}}), \xi(t_{D_{ij}}))$ with a shallower proof, for each $i \in I$ and $j \in J_i$.

It follows from Definition 4.3.1 that there exists a transition rule

$$\rho^c_\xi = \frac{\{t_{P_i} \overset{(\xi(t_{D_i}),l_{ij},\xi(t_{D_{ij}}))}{\longrightarrow} t_{P_{ij}} \mid i \in I, j \in J_i\}}{t_P \overset{(\xi(t_D),l,\xi(t'_D))}{\longrightarrow} t'_P}$$

in the transition rules of $cl(\mathcal{T}^c)$. The induction hypothesis applies to the premises of ρ under σ and ξ, hence we obtain $cl(\mathcal{T}^c) \vdash \sigma(t_{P_i}) \overset{(\xi(t_{D_i}),l,\xi(t_{D_{ij}}))}{\longrightarrow} \sigma(t_{P_{ij}})$ for every $i \in I, j \in J_i$. We, therefore, infer that $cl(\mathcal{T}^c) \vdash \sigma(t_P) \overset{(\xi(t_D),l,\xi(t'_D))}{\longrightarrow} \sigma(t'_P)$, hence $cl(\mathcal{T}^c) \vdash p \overset{(d,l,d')}{\longrightarrow} p'$.

$\Leftarrow$ By induction on the depth of the proof for $cl(\mathcal{T}^c) \vdash p \overset{(d,l,d')}{\longrightarrow} p'$. We dispense with stating the induction basis separately in this case too. Assume that the last transition rule in the proof is

$$\rho' = \frac{\{t_{P_i} \overset{(d_i,l_{ij},d_{ij})}{\longrightarrow} t_{P_{ij}} \mid i \in I, j \in J_i\}}{t_P \overset{(d,l,d')}{\longrightarrow} t'_P};$$

there exists a closed process substitution σ such that $\sigma(t_P) = p$, $\sigma(t'_P) = p'$ and, moreover, it holds that $cl(\mathcal{T}^c) \vdash \sigma(t_{P_i}) \overset{(d_i,l_{ij},d_{ij})}{\longrightarrow} \sigma(t_{P_{ij}})$ with a shallower proof, for each $i \in I$ and $j \in J_i$.

It follows from Definition 4.3.1 that there exists a transition rule

$$\rho = \frac{\{(t_{P_i}, t_{D_i}) \overset{l_{ij}}{\longrightarrow} (t_{P_{ij}}, t_{D_{ij}}) \mid i \in I, j \in J_i\}}{(t_P, t_D) \overset{l}{\longrightarrow} (t'_P, t'_D)}$$

in the transition rules of $\mathcal{T}$ and a data substitution ξ such that $\rho' = \rho^c_\xi$, and hence, $\xi(t_D) = d$, $\xi(t'_D) = d'$, $\xi(t_{D_i}) = d_i$, $\xi(t'_{D_{ij}}) = d'_{ij}$ for every $i \in I, j \in J_i$. The induction hypothesis applies to the premises of ρ', and hence, we obtain $\mathcal{T} \vdash (\sigma(t_{P_i}), \xi(t_{D_i})) \overset{l}{\longrightarrow} (\sigma(t_{P_{ij}}), \xi(t_{D_{ij}})))$ for every $i \in I, j \in J_i$. We therefore infer that $\mathcal{T} \vdash (\sigma(t_P), \xi(t_D)) \overset{l}{\longrightarrow} (\sigma(t'_P), \xi(t'_D))$, hence $\mathcal{T} \vdash (p, d) \overset{l}{\longrightarrow} (p', d')$.

□

Now we are ready to state the proof of Theorem 4.3.1. Let us first prove that $p \leftrightarrows^{\mathcal{T}}_{sl} q$ if, and only if, $p \leftrightarrows^{cl(\mathcal{T}^c)} q$. Again we split the bi-implication into two implications and prove them separately below.

⇒ Since $p \leftrightarrows_{sl}^{\mathcal{T}} q$, there exists a stateless bisimulation relation R w.r.t. $\mathcal{T}$ such that $(p,q) \in R$. We claim that R is also a strong bisimulation relation w.r.t. $cl(\mathcal{T}^c)$. R is symmetric because it is a stateless bisimulation relation and hence, it remains to show that the following transfer condition holds:

If $\mathcal{T}^c \vdash p \xrightarrow{(d,l,d')} p'$, then $cl(\mathcal{T}^c) \vdash q \xrightarrow{(d,l,d')} q'$ for some q' such that $(p',q') \in R$.

It follows from $cl(\mathcal{T}^c) \vdash p \xrightarrow{(d,l,d')} p'$ and the right-to-left implication in Lemma 4.A.1 that $\mathcal{T} \vdash (p,d) \xrightarrow{l} (p',d')$. Since $p \leftrightarrows_{sl}^{\mathcal{T}} q$, we have that $\mathcal{T} \vdash (q,d) \xrightarrow{l} (q',d')$, for some q' such that $(p',q') \in R$. It follows from the latter transition and the left-to-right implication in Lemma 4.A.1 that $cl(\mathcal{T}^c) \vdash q \xrightarrow{(d,l,d')} q'$ and we already had that $(p',q') \in R$, which concludes the proof of the left-to-right implication.

⇐ Symmetric to above. This implication is similar to the informal procedure of turning Modular SOS specifications into SOS specifications in [110, Section 3.9]; there it is mentioned that formalizing the transformation and proving a formal proof of correspondence between the original and the transformed specification is left for future work.

□

4.B The Hybrid Process Algebra HyPA

In [62], a process algebra is presented for the description of hybrid systems, i.e., systems with both discrete events and continuous change of variables. The process signature of HyPA consists of the following process constants and functions:

- process constants: $\mathbf{0}$, ε, $(a)_{a \in A}$, $(c)_{c \in C}$;
- unary process functions: $(init(d)\_)_{d \in D}$, $(\partial_H(\_))_{H \subseteq A}$;
- binary process functions: $\oplus$, $\odot$, $\blacktriangleright$, $\triangleright$, $\|$, $\lfloor\!\lfloor$, and $|$.

We refrain from giving further information about the intended meaning of the sets A, C, and D, and the meaning of the process constants and functions as these are irrelevant to currying. The data state consists of mappings from model variables to values, denoted by Val. The data signature is not made explicit.

The transition system specification defines the following predicate and relations:

- a 'termination'-predicate $\checkmark$;
- a family of 'action-transition' relations $\left(\_ \xrightarrow{l} \_\right)_{l \in A \times Val}$;
- a family of 'flow-transition' relations $\left(\_ \overset{\sigma}{\rightsquigarrow} \_\right)_{\sigma \in T \to Val}$.

Also, the meaning of the set T is irrelevant for our purposes. The 'termination'-predicate is again tacitly considered as transition to a pair of fresh variables. Note that the curried TSS introduces for every predicate a family of predicates for each data term.

The transition rules are given below. Note that in the following semantics, each deduction rule with multiple transition formulae in the conclusion is actually a rule schema representing a separate deduction rule for each and every formula in the conclusion.

$$(1)\ \frac{}{\langle \varepsilon, v\rangle \checkmark}, \qquad (2)\ \frac{}{\langle a, v\rangle \xrightarrow{a,v} \langle \varepsilon, v\rangle}, \qquad (3)\ \frac{}{\langle c, v\rangle \overset{\sigma}{\rightsquigarrow} \langle c, \sigma(t)\rangle}\ {\scriptstyle [(v,\sigma)\models_t c]}_{[dom(\sigma)=[0,t]]},$$

$$(4)\ \frac{\langle x, v'\rangle \checkmark}{\langle init(d)x, v\rangle \checkmark}[(v, v') \models_{\mathrm{r}} d], \qquad (5)\ \frac{\langle x, v'\rangle \xrightarrow{l} \langle y, v''\rangle}{\langle init(d)x, v\rangle \xrightarrow{l} \langle y, v''\rangle}[(v, v') \models_{\mathrm{r}} d],$$

$$(6)\ \frac{\langle x_0, v\rangle \checkmark}{\langle x_0 \oplus x_1, v\rangle \checkmark \quad \langle x_1 \oplus x_0, v\rangle \checkmark}, \qquad (7)\ \frac{\langle x_0, v\rangle \xrightarrow{l} \langle y, v'\rangle}{\langle x_0 \oplus x_1, v\rangle \xrightarrow{l} \langle y, v'\rangle \quad \langle x_1 \oplus x_0, v\rangle \xrightarrow{l} \langle y, v'\rangle},$$

$$(8)\ \frac{\langle x_0, v\rangle \checkmark \quad \langle y_0, v\rangle \checkmark}{\langle x_0 \odot y_0, v\rangle \checkmark}, \qquad (9)\ \frac{\langle x_0, v\rangle \xrightarrow{l} \langle y, v'\rangle}{\langle x_0 \odot x_1, v\rangle \xrightarrow{l} \langle y \odot x_1, v'\rangle},$$

$$(10)\ \frac{\langle x_0, v\rangle \checkmark \quad \langle x_1, v\rangle \xrightarrow{l} \langle y, v'\rangle}{\langle x_0 \odot x_1, v\rangle \xrightarrow{l} \langle y, v'\rangle},$$

$$(11)\ \frac{\langle x_0, v\rangle \checkmark}{\langle x_0 \blacktriangleright x_1, v\rangle \checkmark \quad \langle x_0 \triangleright x_1, v\rangle \checkmark}, \qquad (12)\ \frac{\langle x_0, v\rangle \xrightarrow{l} \langle y, v'\rangle}{\langle x_0 \blacktriangleright x_1, v\rangle \xrightarrow{l} \langle y \blacktriangleright x_1, v'\rangle \quad \langle x_0 \triangleright x_1, v\rangle \xrightarrow{l} \langle y \blacktriangleright x_1, v'\rangle},$$

$$(13)\ \frac{\langle x_1, v\rangle \checkmark}{\langle x_0 \blacktriangleright x_1, v\rangle \checkmark}, \qquad (14)\ \frac{\langle x_1, v\rangle \xrightarrow{l} \langle y, v'\rangle}{\langle x_0 \blacktriangleright x_1, v\rangle \xrightarrow{l} \langle y, v'\rangle},$$

$$(15)\ \frac{\langle x_0, v\rangle \checkmark \quad \langle x_1, v\rangle \checkmark}{\begin{array}{c}\langle x_0 \parallel x_1, v\rangle \checkmark \\ \langle x_0 \mid x_1, v\rangle \checkmark\end{array}}, \qquad (16)\ \frac{\begin{array}{c}\langle x_0, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y_0, v'\rangle \\ \langle x_1, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y_1, v'\rangle\end{array}}{\begin{array}{c}\langle x_0 \parallel x_1, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y_0 \parallel y_1, v'\rangle \\ \langle x_0 \mid x_1, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y_0 \parallel y_1, v'\rangle\end{array}},$$

$$(17)\ \frac{\langle x_0, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle \quad \langle x_1, v\rangle \checkmark}{\begin{array}{c}\langle x_0 \parallel x_1, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle \\ \langle x_1 \parallel x_0, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle \\ \langle x_0 \mid x_1, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle \\ \langle x_1 \mid x_0, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle\end{array}}, \qquad (18)\ \frac{\langle x_0, v\rangle \xrightarrow{a,v'} \langle y, v''\rangle}{\begin{array}{c}\langle x_0 \parallel x_1, v\rangle \xrightarrow{a,v'} \langle y \parallel x_1, v''\rangle \\ \langle x_1 \parallel x_0, v\rangle \xrightarrow{a,v'} \langle x_1 \parallel y, v''\rangle \\ \langle x_0 \between x_1, v\rangle \xrightarrow{a,v'} \langle y \parallel x_1, v''\rangle\end{array}},$$

$$(19)\ \frac{\langle x_0, v\rangle \xrightarrow{a,v'} \langle y_0, v''\rangle \quad \langle x_1, v\rangle \xrightarrow{a',v'} \langle y_1, v''\rangle}{\begin{array}{c}\langle x_0 \parallel x_1, v\rangle \xrightarrow{a'',v'} \langle y_0 \parallel y_1, v''\rangle \\ \langle x_0 \mid x_1, v\rangle \xrightarrow{a'',v'} \langle y_0 \parallel y_1, v''\rangle\end{array}}[a'' = a \gamma a'],$$

$$(20)\ \frac{\langle x, v\rangle \xrightarrow{a,v'} \langle y, v''\rangle}{\langle \partial_H(x), v\rangle \xrightarrow{a,v'} \langle \partial_H(y), v''\rangle}[a \notin H],$$

$$(21)\ \frac{\langle x, v\rangle \overset{\sigma}{\rightsquigarrow} \langle y, v'\rangle}{\langle \partial_H(x), v\rangle \overset{\sigma}{\rightsquigarrow} \langle \partial_H(y), v'\rangle}, \qquad (22)\ \frac{\langle x, v\rangle \checkmark}{\langle \partial_H(x), v\rangle \checkmark}.$$

The curried version of the semantics of HyPA is given below.

$$(1_c)\ \frac{}{\varepsilon \ \checkmark_v}, \qquad (2_c)\ \frac{}{a \overset{(v,(a,v),v)}{\longrightarrow} \varepsilon}, \qquad (3_c)\ \frac{}{c \overset{(v,\sigma,\sigma(t))}{\rightsquigarrow} c}\,{}^{[(v,\sigma)\models_f c]}_{[dom(\sigma)=[0,t]]},$$

$$(4_c)\ \frac{x \ \checkmark_{v'}}{init(d)x \ \checkmark_v}[(v, v') \models_r d], \qquad (5_c)\ \frac{x \overset{(v',l,v'')}{\longrightarrow} y}{init(d)x \overset{\langle v,l,v''\rangle}{\longrightarrow} y}[(v, v') \models_r d],$$

$$(6_c)\ \frac{x_0 \ \checkmark_v}{\begin{array}{c}x_0 \oplus x_1 \ \checkmark_v \\ x_1 \oplus x_0 \ \checkmark_v\end{array}}, \qquad (7_c)\ \frac{x_0 \overset{(v,l,v')}{\longrightarrow} y}{\begin{array}{c}x_0 \oplus x_1 \overset{(v,l,v')}{\longrightarrow} y \\ x_1 \oplus x_0 \overset{(v,l,v')}{\longrightarrow} y\end{array}}, \qquad (8_c)\ \frac{x_0 \ \checkmark_v \quad y_0 \ \checkmark_v}{x_0 \odot y_0 \ \checkmark_v},$$

$$(9_c)\ \frac{x_0 \overset{(\nu,l,\nu')}{\to} y}{x_0 \odot x_1 \overset{(\nu,l,\nu')}{\to} y \odot x_1}, \qquad (10_c)\ \frac{x_0\ \checkmark_\nu \quad x_1 \overset{(\nu,l,\nu')}{\to} y}{x_0 \odot x_1 \overset{(\nu,l,\nu')}{\to} y},$$

$$(11_c)\ \frac{x_0\ \checkmark_\nu}{\begin{array}{c} x_0 \blacktriangleright x_1\ \checkmark_\nu \\ x_0 \triangleright x_1\ \checkmark_\nu \end{array}}, \qquad (12_c)\ \frac{x_0 \overset{(\nu,l,\nu')}{\to} y}{\begin{array}{c} x_0 \blacktriangleright x_1 \overset{(\nu,l,\nu')}{\to} y \blacktriangleright x_1 \\ x_0 \triangleright x_1 \overset{(\nu,l,\nu')}{\to} y \blacktriangleright x_1 \end{array}},$$

$$(13_c)\ \frac{x_1\ \checkmark_\nu}{x_0 \blacktriangleright x_1\ \checkmark_\nu}, \qquad (14_c)\ \frac{x_1 \overset{(\nu,l,\nu')}{\to} y}{x_0 \blacktriangleright x_1 \overset{(\nu,l,\nu')}{\to} y},$$

$$(15_c)\ \frac{x_0\ \checkmark_\nu \quad x_1\ \checkmark_\nu}{\begin{array}{c} x_0 \,\|\, x_1\ \checkmark_\nu \\ x_0 \,|\, x_1\ \checkmark_\nu \end{array}}, \qquad (16_c)\ \frac{\begin{array}{c} x_0 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y_0 \\ x_1 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y_1 \end{array}}{\begin{array}{c} x_0 \,\|\, x_1 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y_0 \,\|\, y_1 \\ x_0 \,|\, x_1 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y_0 \,\|\, y_1 \end{array}},$$

$$(17_c)\ \frac{x_0 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y \quad x_1\ \checkmark_\nu}{\begin{array}{c} x_0 \,\|\, x_1 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y \\ x_1 \,\|\, x_0 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y \\ x_0 \,|\, x_1 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y \\ x_1 \,|\, x_0 \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y \end{array}}, \qquad (18_c)\ \frac{x_0 \overset{(\nu,(a,\nu'),\nu'')}{\to} y}{\begin{array}{c} x_0 \,\|\, x_1 \overset{(\nu,(a,\nu'),\nu'')}{\to} y \,\|\, x_1 \\ x_1 \,\|\, x_0 \overset{(\nu,(a,\nu'),\nu'')}{\to} x_1 \,\|\, y \\ x_0 \,\underline{\|}\, x_1 \overset{(\nu,(a,\nu'),\nu'')}{\to} y \,\|\, x_1 \end{array}}$$

$$(19_c)\ \frac{x_0 \overset{(\nu,(a,\nu'),\nu'')}{\to} y_0 \quad x_1 \overset{(\nu,(a',\nu'),\nu'')}{\to} y_1}{\begin{array}{c} x_0 \,\|\, x_1 \overset{(\nu,(a'',\nu'),\nu'')}{\to} y_0 \,\|\, y_1 \\ x_0 \,|\, x_1 \overset{(\nu,(a'',\nu'),\nu'')}{\to} y_0 \,\|\, y_1 \end{array}} [a'' = a \,\gamma\, a'],$$

$$(20_c)\ \frac{x \overset{(\nu,(a,\nu'),\nu'')}{\to} y}{\partial_H(x) \overset{(\nu,(a,\nu'),\nu'')}{\to} \partial_H(y)} [a \notin H],$$

$$(21_c)\ \frac{x \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} y}{\partial_H(x) \overset{(\nu,\sigma,\nu')}{\rightsquigarrow} \partial_H(y)}, \qquad (22_c)\ \frac{x\ \checkmark_\nu}{\partial_H(x)\ \checkmark_\nu}.$$

On HyPA process terms, in [62], a notion of robust bisimilarity is defined that, for HyPA, coincides with our definition of stateless bisimilarity.

Nondeterministic choice and sequential composition have the same semantics in HyPA as in Linda and hence their algebraic properties follow from an identical line of reasoning. Commutativity of parallel composition follows from the fact the commutative mirror of each rule derived from the rule schemata (15_c), (16_c), (17_c), (18_c), and (19_c) is represented by the same rule schema. Also for parallel composition, all deduction rules but (16_c) and (17_c) are similar to Linda (modulo renaming of labels); deduction rules (16_c) and (17_c) are, respectively, communication and left-choice + testing rules in the Assoc De Simone format of [61] (with the addition of testing operators) and their combination trivially satisfies the constraints of this format (the antecedents of all constraints of this format are false) and hence, associativity of parallel composition follows from Theorem 2 of [61]. Idempotence for $\oplus$ is given because rules (6_c) and (7_c) are choice rules [5, Definition 40] and the family of rules $(6_c), (7_c)$ for all data terms d and d' ensure that the curried specification is in idempotence format with respect to the binary operator $\oplus$. Finding zero and unit elements for the operations $\oplus$, $\odot$, $\blacktriangleright$ and $\triangleright$ is similar to the examples presented in [10].

We derived the following axioms for the semantics of HyPA, using the meta-theorems stated in the third column of the table.

Property	Axiom	Meta-Theorem
Commutativity for $\oplus$	$x_0 \oplus x_1 = x_1 \oplus x_0$	Theorem 4.2.1
Commutativity for $\|\|$	$x_0 \,\|\|\, x_1 = x_1 \,\|\|\, x_0$	Theorem 4.2.1
Associativity for $\oplus$	$x_0 \oplus (x_1 \oplus x_2) = (x_0 \oplus x_1) \oplus x_2$	Theorem 1 of [61]
Associativity for $\odot$	$x_0 \odot (x_1 \odot x_2) = (x_0 \odot x_1) \odot x_2$	Theorem 1 of [61]
Associativity for $\|\|$	$x_0 \,\|\|\, (x_1 \,\|\|\, x_2) = (x_0 \,\|\|\, x_1) \,\|\|\, x_2$	Theorem 2 of [61]
Idempotence for $\oplus$	$x_0 \oplus x_0 = x_0$	Theorem 42 of [5]
Unit element for $\oplus$	$\mathbf{0} \oplus x = x$	Theorem 3 of [10]
Unit element for $\odot$	$\varepsilon \odot x = x$	Theorem 3 of [10]
Unit element for $\blacktriangleright$	$\mathbf{0} \blacktriangleright x = x \blacktriangleright \mathbf{0} = x$	Theorem 3 of [10]
Unit element for $\triangleright$	$x \triangleright \mathbf{0} = x$	Theorem 3 of [10]
Zero element for $\odot$	$\mathbf{0} \odot x = \mathbf{0}$	Theorem 5 of [10]
Zero element for $\triangleright$	$\mathbf{0} \triangleright x = \mathbf{0} \qquad \varepsilon \triangleright x = \varepsilon$	Theorem 5 of [10]
Distributivity of $\odot$ over $\oplus$	$x_0 \oplus x_1 \odot x_2 = x_0 \odot x_1 \oplus x_0 \odot x_2$	Theorem 3 of [9]
Distributivity of $\triangleright$ over $\oplus$	$(x_0 \oplus x_1) \triangleright x_2 = x_0 \triangleright x_2 \oplus x_1 \triangleright x_2$	Theorem 3 of [9]
Distributivity of $\blacktriangleright$ over $\oplus$	$(x_0 \oplus x_1) \triangleright x_2 = x_0 \blacktriangleright x_2 \oplus x_1 \blacktriangleright x_2$	Theorem 3 of [9]
Distributivity of $\partial_H()$ over $\oplus$	$\partial_H(x_0 \oplus x_1) = \partial_H(x_0) \oplus \partial_H(x_1)$	Theorem 3 of [9]
Distributivity of $\partial_H()$ over $\odot$	$\partial_H(x_0 \odot x_1) = \partial_H(x_0) \odot \partial_H(x_1)$	Theorem 3 of [9]
Distributivity of $\partial_H()$ over $\triangleright$	$\partial_H(x_0 \triangleright x_1) = \partial_H(x_0) \triangleright \partial_H(x_1)$	Theorem 3 of [9]
Distributivity of $init()$ over $\oplus$	$init(d)(x_0 \oplus x_1) = init(d)x_0 \oplus init(d)x_1$	Theorem 3 of [9]

Chapter 5

Exploiting Algebraic Laws to Improve Mechanized Axiomatizations

5.1 Introduction

Algebraic properties, such as commutativity, associativity and idempotence of binary operators, specify some natural properties of programming and specification constructs. These properties can either be validated using the semantics of the language with respect to a suitable notion of program equivalence, or they can be guaranteed a priori 'by design'. In particular, for languages equipped with a Structural Operational Semantics (SOS) [94, 123, 124], there are two closely related lines of work to achieve this goal: firstly, there is a rich body of syntactic rule formats that can guarantee the validity of certain algebraic properties; see [23, 118] for recent surveys. Secondly, there are numerous results regarding the mechanical generation of ground-complete axiomatizations of various behavioral equivalences and preorders for SOS language specifications in certain formats—see, e.g., [2, 6, 35, 47, 80, 136].

However, these two lines of research have evolved separately and no link has been established between the two types of results so far. In this chapter, we take the first steps in marrying these two research areas and in using rule formats for algebraic properties (specifically, for commutativity) to enhance the process of automatic generation of axiomatizations for strong bisimilarity from GSOS

language specifications [46, 49]. In particular, we show that linking these two areas results in axiomatizations that look like hand-crafted ones.

Many ground-completeness results have been presented in the literature on process calculi. (See, for instance, the survey paper [11] for pointers to the literature.) A common proof strategy for establishing such ground-completeness results is to reduce the problem of axiomatizing the notion of behavioural equivalence under consideration over arbitrary closed terms to that of axiomatizing it over 'synchronization-tree terms'. This approach is also at the heart of the algorithm proposed in [6] for the automatic generation of finite, equational, ground-complete axiomatizations for bisimilarity over language specifications in the GSOS format. A variation on that algorithm for GSOS language specifications with termination has been presented in [35]. In [136], Ulidowski has instead offered algorithms for the automatic generation of finite axiom systems for the testing preorder over De Simone process languages.

Contribution In Section 5.4 of this chapter, we present a refinement of the algorithm from [6] that uses a rule format guaranteeing commutativity of certain operators to obtain ground-complete axiomatizations of bisimilarity that are closer to the hand-crafted ones than those produced by existing algorithms. (See Section 5.5, where we apply the algorithm to axiomatize the classic parallel composition operator and compare the generated axiomatization to earlier ones.)

Our rule format for commutativity (presented in Section 5.3) is a generalization of the rule format for commutativity from [117], which allows operators to have various sets of commutative arguments. Apart from being natural, such a generalization is useful in the automatic generation of ground-complete axiomatizations, as the developments in this study show.

Structure In Section 5.2 we recall some standard notions from process theory and the meta-theory of SOS. In Section 5.3 we present a generalized notion of commutativity and a corresponding rule format. In Section 5.4 we give an algorithm for the automatic generation of ground-complete axiomatizations for bisimilarity from GSOS language specifications. The algorithm uses information derived from the generalized commutativity format, where possible, to reduce the number of auxiliary operators and to produce axiom systems that are close to hand-crafted ones. In Section 5.5 we apply the algorithm to axiomatize the classic parallel composition operator and compare the generated axiomatization

to earlier ones. Section 5.6 concludes the chapter and suggests some avenues for future research.

This chapter is an extended version of [18]. It offers some explanations, examples and proofs that we needed to remove from the reference in order to conform with the restriction on the number of pages.

5.2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from process theory and the meta-theory of SOS that will be used in the remainder of the chapter. We refer the interested reader to [14, 118] for further details.

5.2.1 Transition System Specifications

Definition 5.2.1 (Signature and terms)**.** *We let V denote an infinite set of variables with typical members $x, x', x_i, y, y', y_i, \ldots$. A* signature Σ *is a set of function symbols, each with a fixed arity. We call these symbols* operators *and usually represent them by $f, g, \ldots$. An operator with arity zero is called a* constant*. We define the set $\mathbb{T}(\Sigma)$ of* terms *over Σ (sometimes referred to as Σ-terms) as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

We use $s, t, t', t_i, u, \ldots$ to range over terms. We write $t_1 \equiv t_2$ if t_1 and t_2 are syntactically equal. The function $vars : \mathbb{T}(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma)$ is the set of closed terms*, i.e., the set of all terms t such that $vars(t) = \emptyset$. We use $p, p', p_i, q, r \ldots$ to range over closed terms. A* substitution *σ is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a* closed substitution*.*

Definition 5.2.2 (Transition System Specifications (TSS), formulae and transition systems)**.** *A* transition system specification *$\mathcal{T}$ is a triple (Σ, L, D) where*

- *Σ is a signature.*

- *L is a set of labels. If $l \in L$ and $t, t' \in \mathbb{T}(\Sigma)$, we say that $t \xrightarrow{l} t'$ is a* positive formula *and $t \xrightarrow{l} \not\to$ is a* negative formula. *A* formula *is either a positive formula or a negative one. A formula is typically denoted by ϕ, ψ, ϕ', ϕ_i, ...*
- *D is a set of* deduction rules, *i.e., pairs of the form (Φ, ϕ) where Φ is a set of formulae and ϕ is a positive formula. We call the formulae contained in Φ the* premises *of the rule and ϕ the* conclusion.

We say that a formula is closed *if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way.*

We often refer to a closed formula $t \xrightarrow{l} t'$ as a *transition* with t being its *source*, l its *label*, and t' its *target*. The notions of source and label are similarly defined for negative formulae. A deduction rule (Φ, ϕ) is typically written as $\frac{\Phi}{\phi}$.

5.2.2 GSOS Format

The GSOS format is a widely studied format of deduction rules in transition system specifications proposed by Bloom, Istrail and Meyer [46, 49]. Transition system specifications whose rules are in the GSOS format enjoy many desirable properties, and several studies in the literature on the meta-theory of SOS have focused on them—see, for instance, [3, 2, 6, 14, 20, 35]. Following [6], in this study we shall also focus on transition system specifications in the GSOS format, which we now proceed to define.

Definition 5.2.3 (GSOS Format [49])**.** *A deduction rule for an operator f of arity n is in the GSOS format if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \xrightarrow{l_{ik}} \not\to \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{l} C[\vec{x}, \vec{y}]}$$

where the x_i's and the y_{ij}'s ($1 \leq i \leq n$ and $1 \leq j \leq m_i$) are all distinct variables, m_i and n_i are natural numbers, $C[\vec{x}, \vec{y}]$ is a Σ-term with variables including at most the x_i's and the y_{ij}'s, and the l_{ij}'s and l are labels. If $m_i > 0$, for some i, then we say that the rule tests its i-th argument positively. Similarly, if $n_i > 0$ then we say that the rule tests its i-th argument negatively.

The above rule is said to be f-defining *and* l-emitting.

A TSS is in the GSOS format when it has a finite signature, a finite set of labels, a finite set of deduction rules and all its deduction rules are in the GSOS format. We shall sometimes refer to a TSS in the GSOS format as a GSOS system.

In addition to the syntactic restrictions on deduction rules, the GSOS format, as presented in [46, 49], requires the signature to include a constant $\mathbf{0}$, a collection of unary operators $a.\_$ ($a \in L$) and a binary operator $\_+\_$. Intuitively, $\mathbf{0}$ represents a process that does not exhibit any behaviour, $s + t$ is the nondeterministic choice between the behaviours of s and t, while $a.t$ is a process that first performs action a and behaves like t afterwards. The standard deduction rules for these operations are given below:

$$\frac{}{a.x_1 \xrightarrow{a} x_1} \qquad \frac{x_1 \xrightarrow{a} x_1'}{x_1 + x_2 \xrightarrow{a} x_1'} \qquad \frac{x_2 \xrightarrow{a} x_2'}{x_1 + x_2 \xrightarrow{a} x_2'}.$$

In the remainder of this chapter, following [46, 49], we shall tacitly assume that each TSS in the GSOS format contains these operators with the rules given above. The import of this assumption is that, as is well known, within each TSS in the GSOS format it is possible to express each finite synchronization tree over L [104]. Following [81], the TSS containing the operators $\mathbf{0}$, $a.\_$ ($a \in L$) and $\_+\_$, with the above-given rules is denoted by BCCSP.

Informally, the intent of a GSOS rule is as follows. Suppose that we are wondering whether $f(\vec{p})$ is capable of taking an l-step. We look at each f-defining and l-emitting rule in turn. We inspect each positive premise $x_i \xrightarrow{l_{ij}} y_{ij}$, checking if p_i is capable of taking an l_{ij}-step for each j and if so calling the l_{ij}-children q_{ij}. We also check the negative premises: if p_i is *in*capable of taking an l_{ik}-step for each k. If so, then the rule *fires* and $f(\vec{p}) \xrightarrow{l} C[\vec{p}, \vec{q}]$. This means that the transition relation $\rightarrow$ associated with a TSS in the GSOS format is the one defined by the rules using structural induction over closed Σ-terms. This transition relation is the unique sound and supported transition relation [82]. Here *sound* means that whenever a closed substitution σ 'satisfies' the premises of a rule of the form given in Definition 5.2.3, then $\sigma(f(x_1, \ldots, x_l)) \xrightarrow{l} \sigma(C[\vec{x}, \vec{y}])$. On the other hand, *supported* means that any transition $p \xrightarrow{l} q$ can be obtained by instantiating the conclusion of a rule of the form given in Definition 5.2.3 with a substitution that satisfies its premises. We refer the interested reader to [46, 49] for the precise definition and much more information on GSOS languages. The above informal description of the transition relation associated with a TSS in the GSOS format suffices to follow the technical developments in the remainder of the chapter.

The notion of disjoint extension of a TSS is defined as in [6].

Definition 5.2.4. *A GSOS system $\mathcal{T}'$ is a* disjoint extension *of a GSOS system $\mathcal{T}$, denoted by $\mathcal{T} \sqsubseteq \mathcal{T}'$, if the signature and rules of $\mathcal{T}'$ include those of $\mathcal{T}$, and $\mathcal{T}'$ introduces no new rules for operators in the signature of $\mathcal{T}$.*

In the light of our assumption, BCCSP $\sqsubseteq \mathcal{T}$ holds for each GSOS system $\mathcal{T}$. If $\mathcal{T}'$ disjointly extends $\mathcal{T}$ then $\mathcal{T}'$ introduces no new outgoing transitions for the closed terms of $\mathcal{T}$. (More general conservative extension results are discussed in, for instance, [70, 113].)

5.2.3 Bisimilarity and Axiom Systems

To establish a link between the operational model and the algebraic properties, a notion of behavioural equivalence should be fixed. The notion of behavioural equivalence that we will use in this chapter is the following, classic notion of bisimilarity [106, 120].

Definition 5.2.5 (Bisimilarity [120])**.** *Let $\mathcal{T}$ be a GSOS system with signature Σ. A relation $R \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* bisimulation *if and only if R is symmetric and, for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ and $l \in L$,*

$$(p_0 \; R \; p_1 \wedge p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (p_1 \xrightarrow{l} p_1' \wedge p_0' \; R \; p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called bisimilar*, denoted by $\mathcal{T} \vdash p_0 \leftrightarroweq p_1$ (or simply by $p_0 \leftrightarroweq p_1$ when $\mathcal{T}$ is clear from the context), when there exists a bisimulation R such that $p_0 \; R \; p_1$.*

It is well known that $\leftrightarroweq$ is an equivalence relation over $\mathbb{C}(\Sigma)$. Any equivalence relation $\sim$ over closed terms in a TSS $\mathcal{T}$ is extended to open terms in the standard fashion, i.e., for all $t_0, t_1 \in \mathbb{T}(\Sigma)$, the equation $t_0 = t_1$ holds over $\mathcal{T}$ modulo $\sim$ (sometimes abbreviated to $t_0 \sim t_1$) if, and only if, $\mathcal{T} \vdash \sigma(t_0) \sim \sigma(t_1)$ for each closed substitution σ.

Remark 5.2.6. *If $\mathcal{T}'$ is a disjoint extension of $\mathcal{T}$, then two closed terms over the signature of $\mathcal{T}$ are bisimilar in $\mathcal{T}$ if and only if they are bisimilar in $\mathcal{T}'$.*

Definition 5.2.7. *Let Σ be a signature. An equivalence relation $\sim$ over Σ-terms is a* congruence *if, for all $f \in \Sigma$ and closed terms $p_1, \ldots, p_n, q_1, \ldots, q_n$, where n is the arity of f, if $p_i \sim q_i$ for each $i \in \{1, \ldots, n\}$ then $f(p_1, \ldots, p_n) \sim f(q_1, \ldots, q_n)$.*

Remark 5.2.8. *Let Σ be a signature and let $\sim$ be a congruence. It is easy to see that, for all $f \in \Sigma$ and terms $t_1, \ldots, t_n, u_1, \ldots, u_n$, where n is the arity of f, if $t_i \sim u_i$ for each $i \in \{1, \ldots, n\}$ then $f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$.*

The following result is well known [46].
Proposition 5.2.9. *$\leftrightarrows$ is a congruence for any TSS in GSOS format.*

Ideally, a notion of behavioural congruence should coincide with the equational theory generated by some 'finitely presentable' set of axioms describing the desired algebraic properties of the operators in a language. One side of this coincidence is captured by the *soundness* requirement, which states that all the (closed) equalities that are derivable from the axiom system using the rules of equational logic are indeed valid with respect to the chosen notion of behavioural equivalence. The other side of the coincidence, called *ground completeness*, states that all the valid behavioural equivalences over *closed* terms are derivable from the axiom system. These concepts are formalized in what follows.
Definition 5.2.10 (Axiom System)**.** *An* axiom system *E over a signature Σ is a set of equalities of the form $t = t'$, where $t, t' \in \mathbb{T}(\Sigma)$. An equality $t = t'$, for some $t, t' \in \mathbb{T}(\Sigma)$, is derivable from E, denoted by $E \vdash t = t'$, if and only if it is in the smallest congruence relation over Σ-terms induced by the equalities in E.*

In the context of a fixed TSS $\mathcal{T}$, an axiom system E (over the same signature) is sound *with respect to a congruence relation $\sim$ if and only if for all $t, t' \in \mathbb{T}(\Sigma)$, if $E \vdash t = t'$, then it holds that $\mathcal{T} \vdash t \sim t'$. The axiom system E is* ground complete *if the implication holds in the opposite direction whenever t and t' are closed terms.*

5.3 Commutativity Format

Commutativity is an essential property specifying that the order of arguments of an operator is immaterial. In the setting of process algebras, commutativity is defined with respect to a notion of behavioural equivalence over terms. In this section, we first present a generalized notion of commutativity that allows n-ary operators to have various sets of commutative arguments and then slightly adapt the commutativity rule format proposed in [117] to the extended setting. Moreover, we give some auxiliary definitions that will be used in the axiomatization procedure proposed in the next section.

In order to motivate the generalized notion of commutativity we present below, consider, by way of example, the ternary operator f defined by the rules below,

where a ranges over the collection of action labels L.

$$\frac{x \xrightarrow{a} x'}{f(x,y,z) \xrightarrow{a} f(x',y,z)} \quad \frac{y \xrightarrow{a} y'}{f(x,y,z) \xrightarrow{a} f(x,y',z)}$$
$$\frac{x \xrightarrow{a} x' \;\; z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')} \quad \frac{y \xrightarrow{a} y' \;\; z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')}.$$

It is not hard to show that the operator f is commutative in its first two arguments modulo bisimilarity, irrespective of the other operators in the TSS under consideration—that is,

$$f(p,q,r) \leftrightarrows f(q,p,r)$$

, for all closed terms p,q,r. On the other hand, the third argument does not commute with respect to the other two. For example, we have that

$$f(a.\mathbf{0},\mathbf{0},\mathbf{0}) \not\leftrightarrows f(\mathbf{0},\mathbf{0},a.\mathbf{0})$$

because $f(a.\mathbf{0},\mathbf{0},\mathbf{0}) \xrightarrow{a} f(\mathbf{0},\mathbf{0},\mathbf{0})$, but $f(\mathbf{0},\mathbf{0},a.\mathbf{0})$ has no outgoing transitions.

The commutativity format presented in [117] can only deal with operators that are commutative for each pair of arguments and, unlike the format that we present below, is therefore unable to detect that f is commutative in its first two arguments.

In what follows, we shall often use $[n]$, $n \geq 0$, to stand for the set $\{1,\ldots,n\}$. Note that $[0]$ is just the empty set.

Definition 5.3.1 (Generalized Commutativity)**.** *Given a set I, a family $\prod_I$ of non-empty, pairwise disjoint subsets of I is called a* partition *of I when $\bigcup \prod_I = I$.*

Let Σ be a signature. Assume that $f \in \Sigma$ is an n-ary operator, $\prod_{[n]}$ is a partition of $[n]$ and $\sim$ is an equivalence relation over $\mathbb{C}(\Sigma)$. The operator f is called $\prod_{[n]}$-commutative with respect to $\sim$ *when, for each $K \in \prod_{[n]}$ and each two $j,k \in K$ such that $j < k$, the following equation is sound with respect to $\sim$:*

$$f(x_1,\ldots,x_n) = f(x_1,\ldots,x_{j-1},x_k,x_{j+1},\ldots,x_{k-1},x_j,x_{k+1},\ldots,x_n).$$

where $x_1,\ldots,x_n$ is a sequence of variables.

Note that the traditional notion of commutativity for binary operators can be recovered using Definition 5.3.1 in terms of $\{\{1,2\}\}$-commutativity. Moreover, the notion of commutativity for n-ary operators from [117] corresponds to $\{[n]\}$-

commutativity. Any n-ary operator is $1_{[n]}$-commutative with respect to any equivalence relation $\sim$, where $1_{[n]} = \{\{1\}, \ldots, \{n\}\}$ is the discrete partition of $[n]$.

From this point onward, whenever a signature Σ is provided, we also assume that every function symbol $f \in \Sigma$ of arity n has an associated fixed partition of its set of arguments $[n]$ denoted by $\prod_f$. We denote the indexed set of all these partitions by $\prod^{\Sigma} = \{\prod_f\}_{f \in \Sigma}$.

Definition 5.3.2. *Let $\prod_1$ and $\prod_2$ be partitions of some set I. We say that $\prod_1$ is* at least as fine as *$\prod_2$ if and only if for each $K_1 \in \prod_1$ there is some $K_2 \in \prod_2$ such that $K_1 \subseteq K_2$. A family of partitions $\prod_1^{\Sigma}$ is at least as fine as $\prod_2^{\Sigma}$ if and only if $\prod_{1f}$ is at least as fine as $\prod_{2f}$, for each $f \in \Sigma$. In other words, this holds if, for each $f \in \Sigma$, the equivalence relation associated with $\prod_{1f}$ is included in the equivalence relation associated with $\prod_{2f}$.*

Definition 5.3.3. *Assume $\Sigma_1 \subseteq \Sigma_2$. Let $\prod^{\Sigma_1}$ be a family of partitions. The* extension of $\prod^{\Sigma_1}$ to Σ_2 *is obtained by taking $\prod_f$ to be the discrete partition over $[n]$ for each $f \in \Sigma_2 \setminus \Sigma_1$, where n is the arity of f.*

Our aim is to define a restriction of the GSOS rule format that guarantees the notion of generalized commutativity defined above for any behavioural equivalence that is coarser than bisimilarity. To this end, we begin by extending the notion of commutative congruence introduced in [117] to the context of this generalized notion of commutativity.

Definition 5.3.4 (Commutative Congruence)**.** *Consider a signature Σ and a set of partitions $\prod^{\Sigma}$. The commutative congruence relation $\sim_{cc}$ (with respect to $\prod^{\Sigma}$) is the least relation over $\mathbb{T}(\Sigma)$ satisfying the following requirements:*

1. *$\sim_{cc}$ is reflexive and transitive;*
2. *$\sim_{cc}$ is a congruence;*
3. *for all $f \in \Sigma$, $K \in \prod_f$, $j, k \in K$ with $j < k$, and $t_1, \ldots, t_n \in \mathbb{T}(\Sigma)$, it holds that*

$$f(t_1, \ldots, t_n) \sim_{cc} f(t_1, \ldots, t_{j-1}, t_k, t_{j+1}, \ldots, t_{k-1}, t_j, t_{k+1}, \ldots, t_n),$$

where n is the arity of f.

Lemma 5.3.5. *$\sim_{cc}$ is an equivalence relation over $\mathbb{T}(\Sigma)$. Moreover, for all terms t, u, if $t \sim_{cc} u$ then $vars(t) = vars(u)$.*

Proof. The relation $\sim_{cc}$ is reflexive and transitive by definition. The fact that it is also symmetric can be shown easily by induction on the definition of $\sim_{cc}$. The second claim also follows by a straightforward induction on the definition of $\sim_{cc}$. □

Lemma 5.3.6. *Let Σ be a signature and let $t, t' \in \mathbb{T}(\Sigma)$. Let σ and σ' be two substitutions. Assume that $t \sim_{cc} t'$ (with respect to some $\prod^{\Sigma}$) and that $\sigma(x) \sim_{cc} \sigma'(x)$, for each $x \in vars(t)$. Then $\sigma(t) \sim_{cc} \sigma'(t')$.*

Proof. The claim can be shown by induction on the definition of $\sim_{cc}$. In the case that $t \sim_{cc} t'$ because $t \equiv t'$, one proceeds by a further induction on the structure of t. The details of the proof are standard and we therefore omit them. □

The following two lemmas can both be shown by induction on the definition of $\sim_{cc}$.

Lemma 5.3.7. *Let Σ be a signature. Assume that $\prod_1^{\Sigma}$ and $\prod_2^{\Sigma}$ are two families of partitions over Σ, and that $\prod_1^{\Sigma}$ is at least as fine as $\prod_2^{\Sigma}$. Then the commutative congruence associated with $\prod_1^{\Sigma}$ is included in the commutative congruence associated with $\prod_2^{\Sigma}$.*

Lemma 5.3.8. *Assume $\Sigma_1 \subseteq \Sigma_2$. Let $\prod^{\Sigma_1}$ be a family of partitions. Then the commutative congruence associated with $\prod^{\Sigma_1}$ over Σ_1-terms is included in the commutative congruence associated with the extension of $\prod_1^{\Sigma_1}$ to Σ_2.*

We are now ready to present a syntactic restriction on the GSOS format that guarantees commutativity with respect to a set of partitions $\prod^{\Sigma}$ modulo any notion of behavioural equivalence that includes strong bisimilarity. Unlike the format for $\{[n]\}$-commutativity given in [117], the format offered below applies to generalized commutativity, in the sense of Definition 5.3.1, and is defined for TSSs whose rules can have negative premises. On the other hand, unlike ours, the format introduced in [117] applies to rules whose positive premises need not have variables as their sources and targets. Extending our format in order to accommodate this kind of premises in deduction rules is straightforward, but is not relevant for the purpose of this chapter.

Definition 5.3.9 (Comm-GSOS)**.** *A transition system specification over signature Σ is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$ if it is in the GSOS format and for each f-defining deduction rule with $f \in \Sigma$ of the following form*

$$\textbf{(d)}\ \frac{H}{f(x_1, \ldots, x_n) \xrightarrow{l} t},$$

each $K \in \prod_f$ and for all $j, k \in K$ with $j < k$, there exist a deduction rule of the following form in the transition system specification

$$\textbf{(d')}\ \frac{H'}{f(x'_1, \ldots, x'_n) \xrightarrow{l} t'}$$

and a bijective mapping $\hbar$ over variables such that

- $\hbar(x'_i) = x_i$ *for each* $i \in [n]$ *such that* $i \neq j$ *and* $i \neq k$,
- $\hbar(x'_j) = x_k$ *and* $\hbar(x'_k) = x_j$,
- $\hbar(t') \sim_{cc} t$, *and*
- $\hbar(H') = H$.

Deduction rule d' is called a commutative mirror *of d (with respect to j, k and $\prod^{\Sigma}$).*

Informally, the role of the bijection $\hbar$ in the definition above is to account for the swapping of variables in the source of the conclusion and a possible bijective renaming of variables. Thus, the above format requires that, when $f \in \Sigma$, for each f-defining rule and for each pair (j,k) of arguments for which f is supposed to be commutative, as specified by $\prod_f$, there exists a commutative mirror that enables the 'same transitions up to the commutative congruence $\sim_{cc}$ associated with $\prod^{\Sigma}$' when the jth and kth arguments of f are swapped. This is the essence of the proof of the following theorem, which states the correctness of the syntactic comm-GSOS format.

Theorem 5.3.10. *If a transition system specification is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$, then each operator $f \in \Sigma$ is $\prod_f$-commutative with respect to any notion of behavioural equivalence that includes bisimilarity.*

Proof. The restriction to closed terms of the commutative congruence relation with respect to $\prod^{\Sigma}$ is a bisimulation. The details of the proof may be found in Section 5.A. □

Remark 5.3.11. *Theorem 5.3.10 would still hold if the last constraint on the bijection $\hbar$ in Definition 5.3.9 were relaxed to $\hbar(H') \subseteq H$.*

Example 5.3.12. *Consider the ternary operator f we used earlier to motivate the notion of generalized commutativity.*

For ease of reference, we recall that the rules for f are

$$\frac{x \xrightarrow{a} x'}{f(x,y,z) \xrightarrow{a} f(x',y,z)} \quad \frac{y \xrightarrow{a} y'}{f(x,y,z) \xrightarrow{a} f(x,y',z)}$$
$$\frac{x \xrightarrow{a} x' \quad z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')} \quad \frac{y \xrightarrow{a} y' \quad z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')},$$

where a ranges over L.

Any transition system specification including the operator f is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_f = \{\{1,2\},\{3\}\}$. Indeed, the a-emitting rules in the first row are one the commutative mirror of the other with respect to $\prod^{\Sigma}$, and so are those in the second row.By way of example, consider the rule $\dfrac{y \xrightarrow{a} y' \quad z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')}$ *and take $j = 1$ and $k = 2$ in Definition 5.3.9. To see that the rule* $\dfrac{x \xrightarrow{a} x' \quad z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')}$ *is a commutative mirror of the other with respect to $\prod^{\Sigma}$, take the bijection $\hbar$ over variables such that $\hbar(x) = y, \hbar(y) = x, \hbar(x') = y', \hbar(y') = x'$ and that is the identity function on all the other variables. Then $\hbar(\{x \xrightarrow{a} x',\ z \xrightarrow{a} z'\}) = \{y \xrightarrow{a} y',\ z \xrightarrow{a} z'\}$ and $\hbar(f(x',y,z')) = f(y',x,z') \sim_{cc} f(x,y',z')$. The constraints in Definition 5.3.9 are vacuously satisfied when we take $K = \{3\}$.*

Therefore, by Theorem 5.3.10, we recover the fact that f is commutative in its first two arguments.

Example 5.3.13 (Parallel Composition)**.** *A frequently occurring commutative operator is parallel composition. It appears in, amongst others, ACP [44], CCS [106], and CSP [95]. Here we discuss parallel composition with communication in the style of ACP [44]. The rules for this operator are listed below. In those rules, a, b, c range over L and $\gamma : L \times L \hookrightarrow L$ is a partial communication function.*

$$(\mathbf{p}_1)\ \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad (\mathbf{p}_2)\ \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad (\mathbf{p}_3)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \parallel y \xrightarrow{c} x' \parallel y'}\ \gamma(a,b) = c$$

If the partial communication function γ is commutative, then any GSOS system including the operator $\parallel$ given by the above rules is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{\parallel} = \{\{1,2\}\}$. Hence it follows from Theorem 5.3.10 that $\parallel$ is $\{\{1,2\}\}$-commutative.

Example 5.3.14 (Timed Alternative Composition)**.** *Below, we consider the timing rules for a discrete-time version of the alternative composition operator [29]. This operator is commutative and its defining rules involve negative premises. (In the rules below, 1 denotes the discrete time transition.)*

$$(\mathbf{at}_1)\ \frac{x \xrightarrow{1} x' \quad y \overset{1}{\nrightarrow}}{x \otimes y \xrightarrow{1} x'} \qquad (\mathbf{at}_2)\ \frac{x \overset{1}{\nrightarrow} \quad y \xrightarrow{1} y'}{x \otimes y \xrightarrow{1} y'} \qquad (\mathbf{at}_3)\ \frac{x \xrightarrow{1} x' \quad y \xrightarrow{1} y'}{x \otimes y \xrightarrow{1} x' \otimes y'}$$

Any TSS including the operator $\otimes$ given by the above rules is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{\otimes} = \{\{1,2\}\}$. Indeed, with respect to such $\prod^{\Sigma}$, each of the rules $(\mathbf{at}_1)$ *and* $(\mathbf{at}_2)$ *is the commutative mirror of the other, and rule*

(at$_3$) *is its own commutative mirror. From Theorem 5.3.10, it follows that* $\otimes$ *is* $\{\{1,2\}\}$*-commutative. (By including the standard action transitions for alternative composition, such as those given for the binary operator* $+$*, the same result can be obtained.)*

Lemma 5.3.15. *Assume that a TSS* $\mathcal{T}$ *is in the comm-GSOS format with respect to a family of partitions* $\prod_1^\Sigma$*, and* $\prod_1^\Sigma$ *is at least as fine as* $\prod_2^\Sigma$*. Then* $\mathcal{T}$ *is also in the comm-GSOS format with respect to* $\prod_2^\Sigma$*.*

Proof. The claim follows from Lemma 5.3.7. □

5.4 Mechanized Axiomatization

In this section, we present a technique for the automatic generation of ground-complete axiomatizations of bisimilarity over TSSs in the comm-GSOS format, which is derived from the one introduced in [6]. Our approach improves upon the one in [6] by making use of the rule format for generalized commutativity we introduced in the previous section. Our goal is to generate a disjoint extension of the original TSS and a finite axiom system that is sound and ground complete for bisimilarity over it. This finite axiom system may then also be used for equationally establishing bisimilarity over closed terms from the original TSS. We start by axiomatizing a rather restrictive subset of 'good' operators in Section 5.4.1. Then we turn 'bad' operators into good ones by means of auxiliary operators. In both of these steps, we exploit commutativity information, where possible, in order to reduce the number of generated axioms, as well as the number of generated auxiliary operators.

5.4.1 Axiomatizing Good Operators

The approach offered in [6] relies on the fact that the signature includes the three operators whose semantics has been presented in Section 5.2.2, namely the deadlock constant, the prefixing operator and the nondeterministic choice operator. (Recall that, in keeping with [46, 49], we assume that these operators are present in any TSS in the GSOS format.)

Definition 5.4.1. *A term* t *is in* head normal form *if it has the form* $a_1.t_1 + \cdots + a_n.t_n$ *for some* $n \geq 0$*, some set of actions* $\{a_i \mid i \in [n]\}$ *and set of terms* $\{t_i \mid i \in [n]\}$*. If* $n = 0$ *then* $a_1.t_1 + \cdots + a_n.t_n$ *stands for* $\mathbf{0}$*.*

The aim of the axiomatization procedure is then to generate an axiom system that can rewrite any closed term p into a term p' in head normal form such that $p \leftrightarrows p'$. (We call an axiom system with this property *head normalizing*.)

For 'semantically well founded' terms (see [6, Definition 5.1 on page 28]), rewriting into head normal form can be used to prove that each closed term is equal to a closed term over the signature for BCCSP. This leads to a ground-complete axiomatization of bisimilarity, since BCCSP is finitely axiomatized modulo bisimilarity by the following axiom system E_{BCCSP} from [92]:

$$\begin{array}{ll} x + y = y + x & x + x = x \\ (x + y) + z = x + (y + z) & x + \mathbf{0} = x. \end{array}$$

Proposition 5.4.2 (Hennessy and Milner [92])**.** *The axiom system E_{BCCSP} is sound and ground complete for bisimilarity over BCCSP.*

To start with, we focus on the case of closed terms built using only *good* operators, which we now proceed to define.

Definition 5.4.3 (Smooth and distinctive operator)**.** *Consider an n-ary operator f.*

1. *A* smooth GSOS deduction rule *is of the form*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\} \cup \{x_i \overset{b_{ij}}{\nrightarrow} \mid i \in J, 1 \leq j \leq n_i\}}{f(x_1, \ldots, x_n) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

   where

   (a) *I and J are disjoint subsets of $[n]$ such that $I \cup J = [n]$;*

   (b) *$C[\vec{x}, \vec{y}]$ can only include the variables x_i ($i \in [n] \setminus I$) and y_i ($i \in I$).*

   An operator f of a TSS in the GSOS format is smooth *if all its rules are smooth.*

2. *An n-ary operator f of a TSS in the GSOS format is* distinctive *if it is smooth, each f-defining rule tests the same set of arguments I positively, and for every two distinct f-defining rules there is some argument tested positively by both rules, but with a different action.*

We refer the interested reader to [6, Section 4.1] for an in-depth discussion of the constraints for smooth and distinctive operators.

Remark 5.4.4. *The ternary operator f from Example 5.3.12 and the parallel composition operator from Example 5.3.13 are smooth but not distinctive. On the other hand, the*

classic communication merge operator [25, 43], given by the rules

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \mid y \xrightarrow{c} x' \parallel y'} (\gamma(a,b) = c),$$

is smooth and distinctive. Moreover, assuming that γ is commutative, any TSS whose signature Σ includes $\parallel$ and $\mid$ with the previously given rules is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{\mid} = \prod_{\parallel} = \{\{1,2\}\}$.

Definition 5.4.5 (Discarding and Good Operators)**.** *A smooth GSOS rule of the form given in Definition 5.4.3 is* discarding *if none of the variables x_i with $i \in J$ and $n_i > 0$ occurs in $C[\vec{x}, \vec{y}]$. A smooth operator is* discarding *if so are all the rules for it. A smooth operator is* good *[53] if it is both distinctive and discarding.*

In the remainder of this subsection, we assume that the GSOS system $\mathcal{T}$ has signature Σ and is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$. Let $f \in \Sigma$ be a good operator that is not in the signature for BCCSP, and let n be its arity. Our goal is to generate an axiom system that can be used to turn any term of the form $f(t_1, \ldots, t_n)$, where the t_i's are in head normal form, into a head normal form. In the generation of the axiom system, we will exploit the commutativity information that is provided by the partition $\prod_f$ and therefore we assume that $n \geq 2$. (If f is either a constant or a unary operator, then it will be axiomatized exactly as in [6], since commutativity information is immaterial.) Let $I_f \subseteq [n]$ be the set of arguments that are tested positively by f and let J_f be the complement of I_f.

Assume that $\prod_f = \{K_1, \ldots, K_\ell\}$. Since $\mathcal{T}$ is in the comm-GSOS format with respect to $\prod^{\Sigma}$, and f is smooth and distinctive, it is not hard to see that for each $h \in [\ell]$,

$$K_h \subseteq I_f \text{ or } K_h \subseteq J_f.$$

Indeed exactly one of the above inclusions holds. Let $\prod_f^+ = \{K \mid K \in \prod_f \text{ and } K \subseteq I_f\}$ and $\prod_f^- = \{K \mid K \in \prod_f \text{ and } K \subseteq J_f\}$. We use K_f^+ (respectively, K_f^-) to denote a subset of I_f (respectively, J_f) that results by choosing exactly one representative element for each $K \in \prod_f^+$ (respectively, $K \in \prod_f^-$).

Example 5.4.6. *Consider the communication merge operator $\mid$ from Remark 5.4.4. We already remarked that, when the communication function γ is commutative, the rules for $\mid$ are in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{\mid} = \prod_{\parallel} = \{\{1,2\}\}$. For the operator $\mid$, we may take $K_{\mid}^+ = \{1\}$. Since the rules for $\mid$ have no negative premises, $K_{\mid}^-$ is empty.*

Definition 5.4.7. *Let f be a good n-ary operator, and let K_f^+ and K_f^- be defined as above. We associate with f the finite axiom system E_f consisting of the following equations.*

1. Distributivity laws*: For each $i \in K_f^+$, we have the equation:*

$$f(x_1, \ldots, x_i + x_i', \ldots, x_n) = f(x_1, \ldots, x_i, \ldots, x_n) + f(x_1, \ldots, x_i', \ldots, x_n).$$

2. Peeling laws*: For each rule for f of the form given in Definition 5.4.3, each $k \in K_f^-$ with $n_k > 0$ and each $a \notin \{b_{kj} \mid 1 \leq j \leq n_k\}$, we have the equation:*

$$f(P_1, \ldots, P_n) = f(Q_1, \ldots, Q_n),$$

where $P_i \equiv \begin{cases} a_i.y_i & i \in I \\ a.x_k' + x_k'' & i = k \\ x_i & \text{otherwise} \end{cases}$ *and* $Q_i \equiv \begin{cases} a_i.y_i & i \in I \\ x_k'' & i = k \\ x_i & \text{otherwise.} \end{cases}$

3. Action laws*: For each rule for f of the form given in Definition 5.4.3, we have the equation:*

$$f(P_1, \ldots, P_n) = c.C[\vec{P}, \vec{y}],$$

where

$$P_i \equiv \begin{cases} a_i.y_i & i \in I \\ \mathbf{0} & i \in J \text{ and } n_i > 0 \\ x_i & \text{otherwise.} \end{cases}$$

4. Inaction laws*: For each $i \in K_f^+$, we have the equation*

$$f(x_1, \ldots, x_{i-1}, \mathbf{0}, x_{i+1}, \ldots, x_n) = \mathbf{0}.$$

Suppose that, for each $i \in [n]$, term P_i is of the form $a.z_i$ when $i \in I_f$, and of the form $a.z_i + z_i'$ or z_i when $i \in J_f$. Suppose further that, for each rule for f of the form given in Definition 5.4.3, there exists some $i \in [n]$ such that one of the following holds:

- *$i \in I_f$ and ($P_i \equiv a.z_i$, for some $a \neq a_i$),*
- *$i \in J_f$ and ($P_i \equiv b_{ij}.z_i + z_i'$, for some $1 \leq j \leq n_i$).*

Then we have the equation $f(P_1, \ldots, P_n) = \mathbf{0}$.

5. Commutativity laws*: For each equivalence class $K \in \prod_f$ and each two $i, j \in K$ such that $i < j$, we have the equation:*

$$f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n).$$

The axiom system E_f (a rewrite system, when the axioms are oriented from left to right), in combination with the axioms for BCCSP, can be used to turn any term of the form $f(t_1, \ldots, t_n)$, where the t_i's are in head normal form, into a head normal form.

Theorem 5.4.8. *Consider a TSS $\mathcal{T}$ in GSOS format. Let Σ_g be a collection of good operators of $\mathcal{T}$. Let E_{Σ_g} be the finite axiom system that consists of the axioms in E_{BCCSP} and the axioms in E_f, for each $f \in \Sigma_g$.*

The following statements hold for any GSOS system $\mathcal{T}'$ such that $\mathcal{T} \sqsubseteq \mathcal{T}'$:

1. *E_{Σ_g} is sound modulo bisimilarity over $\mathbb{T}(\Sigma')$, where Σ' is the signature of $\mathcal{T}'$.*
2. *E_{Σ_g} is complete for terms built using the operations in the signature Σ_g and those in the signature of BCCSP.*

Proof. The two statements can be shown following the lines of the corresponding theorem in [6]. See also the proof of Theorem 5.8 in [1]. □

Example 5.4.9. *For the communication merge operator $|$, taking $K_{|}^{+} = \{1\}$ as in Example 5.4.6, Definition 5.4.7 yields the following axiom system $E_{|}$:*

$$
\begin{array}{lrcl}
\textit{distributivity:} & (x + y) \mid z & = & (x \mid z) + (y \mid z), \\
\textit{action:} & a.x \mid b.y & = & c.(x \parallel y) \quad \textit{if } \gamma(a, b) = c, \\
\textit{inaction:} & \mathbf{0} \mid y & = & \mathbf{0}, \\
\textit{inaction:} & a.x \mid b.y & = & \mathbf{0} \quad \textit{if } \gamma(a, b) \textit{ is undefined}, \\
\textit{commutativity:} & x \mid y & = & y \mid x.
\end{array}
$$

These are exactly the equations describing the interplay between the operator $|$ and the BCCSP operators given in Table 7.1 on page 204 of [25].

5.4.2 Turning Bad into Good

In order to handle arbitrary GSOS operators, one needs two additional procedures: one for transforming non-smooth operators into smooth and discarding (but not necessarily distinctive) operators, and one for expressing smooth, discarding and non-distinctive operators in terms of good operators. We adopt the same approach for the first procedure as the one presented in Lemma 4.13 in [6]. On the other hand, for the second procedure, we improve on the algorithm derived from Lemma 4.10 in [6].

The step from smooth, discarding and non-distinctive operators to good ones involves the synthesis of several new operators. We now show how to improve this transformation, as presented in the aforementioned reference, by reducing the number of the generated auxiliary operators, making use of the ideas underlying the generalized commutativity format presented in Section 5.3.

Making Smooth and Discarding Operators Distinctive. Consider a TSS $\mathcal{T}$ with signature Σ in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$. Let $f \in \Sigma$ be a smooth and discarding, but not distinctive operator, and let n be its arity. We will now show how to express f in terms of good operators. We start with partitioning the set of f-defining rules into sets $R_1, \ldots, R_m$, $m > 1$, such that f is distinctive when its rules are restricted to those in R_i for each $i \in [m]$. Note that all the rules in each R_i test the same arguments positively. If $\prod_f$ is the discrete partition over $[n]$ then one proceeds by axiomatizing f as in the version of the original algorithm based on the so-called peeling laws presented in [6]. Indeed, in that case, f has no pair of commutative arguments. Suppose therefore that $\prod_f$ is not the discrete partition, and take some $K \in \prod_f$ of maximum cardinality.

Remark 5.4.10. *Any non-singleton K would do in what follows. However, picking a K of maximum cardinality will reduce the number of auxiliary operators that is generated by the procedure outlined below.*

Our aim now is to define when two sets of rules for f are 'essentially the same up to the commutative arguments in K' and to use this information in order to synthesize enough good operators for expressing f up to bisimilarity.

Definition 5.4.11.

- *Let d and d' be two f-defining and l-emitting rules. We say that d' is a* commutative mirror *of d with respect to K and $\prod^{\Sigma}$ if the constraints in Definition 5.3.9 are met for some $j, k \in K$ with $j < k$.*
- *We use $\overset{K}{\smile}$ to denote the reflexive and transitive closure of the relation 'is a commutative mirror with respect to K'.*
- *Let R and R' be two sets of f-defining rules. We write $R \overset{K}{\smile} R'$ if, and only if,*
  - *for each $d \in R$ there is some $d' \in R'$ such that $d \overset{K}{\smile} d'$, and*
  - *for each $d' \in R'$ there is some $d \in R$ such that $d \overset{K}{\smile} d'$.*

Example 5.4.12. *Consider the ternary operator f defined by the rules on page 110. That operator is smooth and discarding, but not distinctive. Collecting all the rules that test the same arguments positively in the same set, we obtain the following four sets of rules:*

- R_1 *contains all the rules of the form* $\dfrac{x \xrightarrow{a} x'}{f(x,y,z) \xrightarrow{a} f(x',y,z)}$ $(a \in L)$.
- R_2 *contains all the rules of the form* $\dfrac{y \xrightarrow{a} y'}{f(x,y,z) \xrightarrow{a} f(x,y',z)}$ $(a \in L)$.
- R_3 *contains all the rules of the form* $\dfrac{x \xrightarrow{a} x',\ z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')}$ $(a \in L)$.
- R_4 *contains all the rules of the form* $\dfrac{y \xrightarrow{a} y',\ z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')}$ $(a \in L)$.

We have already seen in Example 5.3.12 that any GSOS system including the operator f is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_f = \{\{1,2\},\{3\}\}$. Take $K = \{1,2\}$.

It is not hard to see that $R_1 \overset{K}{\smile} R_2$ and $R_3 \overset{K}{\smile} R_4$ hold. Indeed, as we observed in Example 5.3.12, each a-emitting rule in R_1 (respectively, R_3) is the commutative mirror of the a-emitting rule in R_2 (respectively, R_4) with respect to K, and vice versa.

Lemma 5.4.13. *$\overset{K}{\smile}$ is an equivalence relation over f-defining rules and over sets of f-defining rules.*

The following notion will be useful in characterizing the relation $\overset{K}{\smile}$ over the collection of f-defining rules.

Definition 5.4.14. *Let $n > 0$ and let $K \subseteq [n]$. A bijection $\pi : [n] \to [n]$ is a K-*permutation *if it is the identity function over $[n] \setminus K$.*

Lemma 5.4.15. *Let d and d' be two f-defining and l-emitting rules. Suppose that $d = \dfrac{H}{f(x_1,\ldots,x_n) \xrightarrow{l} t}$ and $d' = \dfrac{H'}{f(x_1,\ldots,x_n) \xrightarrow{l} t'}$. Then $d \overset{K}{\smile} d'$ if, and only if there are some K-permutation π and some bijection $\hbar$ over variables such that*

- $\hbar(x_i) = x_{\pi(i)}$, *for each $i \in [n]$,*
- $\hbar(t') \sim_{cc} t$, *and*
- $\hbar(H') = H$.

Proof. By induction on the definition of $\overset{K}{\smile}$. □

Recall that $\{R_1,\ldots,R_m\}$, $m > 1$, is a partition of the set of f-defining rules such that f is distinctive when its rules are restricted to those in R_i for each $i \in [m]$. Consider $\{R_1,\ldots,R_m\}/\overset{K}{\smile}$, the quotient of the set $\{R_1,\ldots,R_m\}$ with respect to the equivalence relation $\overset{K}{\smile}$. Let $\rho_1,\ldots,\rho_\ell$ be representatives of its equivalence classes. For example, in the case of the operator considered in Example 5.4.12 above, one

could pick R_1 and R_4, say, as representatives of the two equivalence classes with respect to $\overset{\{1,2\}}{\smile}$.

We proceed by adding to the signature Σ fresh n-ary operator symbols $f_1, \ldots, f_\ell$. The rules for the operator f_i are obtained by simply turning those in ρ_i into f_i-defining ones. Let $\mathcal{T}'$ be the resulting disjoint extension of $\mathcal{T}$. Following [6], we now need to generate an axiom that expresses f in terms of $f_1, \ldots, f_\ell$.

Definition 5.4.16. *Let $n > 0$ and let $K \subseteq [n]$. A bijection $\pi : [n] \to [n]$ is a K-*permutation *if it is the identity function over $[n] \setminus K$.*

The equation relating f to the f_i's, $i \in [\ell]$, can now be stated as follows:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{\ell} \sum \{f_i(x_{\pi(1)}, \ldots, x_{\pi(n)}) \mid \pi \text{ is a } K\text{-permutation}\}. \tag{5.1}$$

For our running example, namely the ternary operator f defined by the rules on page 110 and considered in Examples 5.3.12 and 5.4.12, with the choice of representatives mentioned above, there are two auxiliary operators f_1 and f_2 with rules $\dfrac{x \xrightarrow{a} x'}{f_1(x,y,z) \xrightarrow{a} f(x',y,z)}$ $\dfrac{y \xrightarrow{a} y',\ z \xrightarrow{a} z'}{f_2(x,y,z) \xrightarrow{a} f(x,y',z')}$, where a ranges over L. Apart from the identity function over [3], the only $\{1,2\}$-permutation is the one that swaps 1 and 2. Therefore, instantiating equation (5.1), we obtain that

$$f(x_1,x_2,x_3) = f_1(x_1,x_2,x_3) + f_1(x_2,x_1,x_3) + f_2(x_1,x_2,x_3) + f_2(x_2,x_1,x_3).$$

Using Definition 5.3.3, the family of partitions $\prod^{\Sigma}$ can be extended to any signature that includes the signature $\Sigma \cup \{f_i \mid i \in [\ell]\}$. Note that any disjoint extension of $\mathcal{T}'$ is in the comm-GSOS format with respect to this extension of $\prod^{\Sigma}$.

Proposition 5.4.17. *Equation (5.1) is sound in any disjoint extension of $\mathcal{T}'$.*

Proof. See Appendix 5.B. □

Equation (5.1) can be simplified in case any of the auxiliary operators $f_1, \ldots, f_\ell$ is commutative in the set of arguments K. Indeed, let $N \subseteq [\ell]$, and assume that $\mathcal{T}'$ is in the comm-GSOS format with respect to the family of partitions that associates with each operator g the partition $\prod_g^{\Sigma}$ when $g \in \Sigma$, the partition $\{K\} \cup 1_{[n]\setminus K}$ when $g \in \{f_i \mid i \in N\}$, and the partition $1_{[n]}$ otherwise. Then we have the following result.

Proposition 5.4.18. *The following equation is sound in any disjoint extension of $\mathcal{T}'$.*

$$f(x_1,\ldots,x_n) = \sum_{i\in[\ell]\setminus N}\sum\{f_i(x_{\pi(1)},\ldots,x_{\pi(n)}) \mid \pi \textit{ is a K-permutation}\} + \sum_{i\in N} f_i(x_1,\ldots,x_n) \tag{5.2}$$

In the following section, we will see that the above simplification leads to an axiomatization of the classic parallel composition operator that is equal to an existing hand-crafted one. Of course, if either N or $[\ell]\setminus N$ are empty, the corresponding $\mathbf{0}$ summand can be omitted in equation (5.2).

Turning Non-Smooth Operators into Smooth Ones. The methods we have presented so far yield an algorithm that, given a TSS $\mathcal{T}$ with signature Σ in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$, can be used to generate a disjoint extension $\mathcal{T}'$ of $\mathcal{T}$ over some signature Σ' that includes Σ and a finite axiom system E such that E is sound modulo bisimilarity over any disjoint extension of $\mathcal{T}'$ and is head normalizing for all closed Σ'-terms. Ground-completeness of E with respect to bisimilarity over $\mathcal{T}'$ (and therefore over $\mathcal{T}$) follows using standard reasoning, by possibly using the well-known Approximation Induction Principle [43] if $\mathcal{T}'$ is not semantically well founded. See [6] for details.

The algorithm has the following steps:

1. Start with the axiom system E_{BCCSP} and consider next the operators that are not in the signature for BCCSP.
2. For each non-smooth operator $f \in \Sigma$, generate a fresh smooth and discarding operator f', and add to the axiom system the equation expressing f in terms of f' as in Lemma 4.13 in [6].
3. For each smooth and discarding, but not distinctive, operator f in the resulting signature, generate a family of fresh good operators $f_1,\ldots f_\ell$, as indicated in this section, and add to the axiom system the instance of equation (5.1) or of equation (5.2), as appropriate, expressing f in terms of $f_1,\ldots f_\ell$.
4. For each good operator in the resulting signature, add to the axiom system the equations mentioned in the statement of Theorem 5.4.8.

5.5 Axiomatizing Parallel Composition

Let us concretely analyze the axiomatization derived using the procedure described above for the classic parallel composition operator $\|$ from Example 5.3.13.

We assume henceforth that the partial synchronization function γ is commutative, so that $\|$ is $\{\{1,2\}\}$-commutative. As we observed in Remark 5.4.4, the parallel composition operator is smooth but not distinctive. When we partition the set of rules for $\|$ into subsets of rules that test the same arguments positively, we obtain three sets R_1, R_2 and R_3, where each R_i consists of all the instances of rule (p_i) from Example 5.3.13. It is easy to see that $R_1 \overset{\{1,2\}}{\smile} R_2$. Therefore, following the procedure described in Section 5.4.2, we can generate two auxiliary binary operators, which are the classic left merge and communication operators, denoted by $\between$ and $|$, respectively. The rules for $|$ are those in Remark 5.4.4 and those for the left merge operator are

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \| y} \ (a \in L).$$

Since we know that $|$ is $\{\{1,2\}\}$-commutative, the relationship between $\|$ and the two auxiliary operators can be expressed using equation (5.2), whose relevant instance becomes

$$x \| y = (x \between y) + (y \between x) + (x \mid y).$$

This is exactly equation M in Table 7.1 on page 204 of [25]. The axioms for $|$ produced by our methods are those given in Example 5.4.9. On the other hand, the left merge operator is axiomatized as in [6] since commutativity information is immaterial for it.

In Figure 5.1 we compare the axiomatization for the parallel composition operator $\|$ derived using the algorithm from [6] and the 'optimized axiomatization' one obtains using the algorithm mentioned above. (We omit the four equations in the axiom system E_{BCCSP} recalled in Section 5.4.) The axioms generated by the algorithm from [6] do resemble the original axioms of [44] to a large extent. The auxiliary operator $\between$ is called right merge in the literature.

Standard			Optimized		
$x \parallel y$	$=$	$(x \mathbin{\underline{\parallel}} y) + (x \mathbin{\underline{\parallel}} y) + (x \mid y)$	$x \parallel y$	$=$	$(x \mathbin{\underline{\parallel}} y) + (y \mathbin{\underline{\parallel}} x) + (x \mid y)$
$(a.x) \mathbin{\underline{\parallel}} y$	$=$	$a.(x \parallel y)$	$(a.x) \mathbin{\underline{\parallel}} y$	$=$	$a.(x \parallel y)$
$x \mathbin{\underline{\parallel}} (a.y)$	$=$	$a.(x \parallel y)$	$(a.x) \mid (b.y)$	$=$	$c.(x \parallel y)$ if $\gamma(a,b) = c$
$(a.x) \mid (b.y)$	$=$	$c.(x \parallel y)$ if $\gamma(a,b) = c$	$(x + y) \mathbin{\underline{\parallel}} z$	$=$	$(x \mathbin{\underline{\parallel}} z) + (y \mathbin{\underline{\parallel}} z)$
$(x + y) \mathbin{\underline{\parallel}} z$	$=$	$(x \mathbin{\underline{\parallel}} z) + (y \mathbin{\underline{\parallel}} z)$	$(x + y) \mid z$	$=$	$(x \mid z) + (y \mid z)$
$x \mathbin{\underline{\parallel}} (y + z)$	$=$	$(x \mathbin{\underline{\parallel}} y) + (x \mathbin{\underline{\parallel}} z)$			
$(x + y) \mid z$	$=$	$(x \mid z) + (y \mid z)$			
$x \mid (y + z)$	$=$	$(x \mid y) + (x \mid z)$	$\mathbf{0} \mathbin{\underline{\parallel}} x$	$=$	$\mathbf{0}$
$\mathbf{0} \mathbin{\underline{\parallel}} x = \mathbf{0}$		$x \mathbin{\underline{\parallel}} \mathbf{0} = \mathbf{0}$	$\mathbf{0} \mid x$	$=$	$\mathbf{0}$
$\mathbf{0} \mid x = \mathbf{0}$		$x \mid \mathbf{0} = \mathbf{0}$	$(a.x) \mid (b.y)$	$=$	$\mathbf{0}$ if $\gamma(a,b)$ is undefined
$(a.x) \mid (b.y)$	$=$	$\mathbf{0}$ if $\gamma(a,b)$ is undefined	$x \mid y$	$=$	$y \mid x$

Figure 5.1: Axiomatizing $\parallel$

We implemented the axiomatizations in the rewriting logic specification and programming language Maude [59]. The optimized axiomatization consists of fewer axioms and uses only two auxiliary operators instead of three, yet it brings terms into normal forms, which only contain the operators from the signature of BCCSP, only negligibly faster than the standard axiomatization. For example, the term $a^3 \parallel b^3 \parallel c^3$ is reduced to its normal form by performing 5736 rewrites using the optimized axiomatization and 5748 when using the standard one, at the same average speed of ~350.000 rewrites/second. For the purpose of the experiment, we considered the communication function defined by $\gamma(l_1, l_2) = \min(l_1, l_2)$, where $l_1, l_2 \in \{a, b, c\}$ and $a < b < c$.

5.6 Conclusions and Future Work

In this chapter, we have taken a first step towards marrying two lines of development within the field of the meta-theory of SOS, viz. the study of algorithms for the automatic generation of ground-complete axiomatizations for bisimilarity from SOS specifications (see, for instance, [6, 35, 136]) and the development of rule formats guaranteeing the validity of algebraic laws, such as those surveyed in [23]. More specifically, we have presented a rule format for commutativity that refines the one offered in [117] in that it allows one to consider various sets of commutative arguments, and we have used the information provided by that rule format to refine the algorithm for the automatic generation of ground-complete

axiomatizations for bisimilarity from [6]. The resulting procedure yields axiom systems that use fewer auxiliary operators to axiomatize commutative operators than the one from [6]. Moreover, in some important cases, the mechanically produced axiomatizations of some operators are identical to the hand-crafted ones from the literature.

To the best of our knowledge, the ideas we have presented in this chapter have never been explored before, and they enrich the toolbox one can use when reasoning about bisimilarity by means of axiomatizations. Moreover, the combination of two closely related strands of research on the meta-theory of SOS we have begun in this chapter is of theoretical interest and may lead to further improvements on algorithms for the automatic generation of axiomatic characterizations of bisimilarity. As future work, we will implement the axiomatization procedure presented in this chapter in the PREG Axiomatizer tool (see Chapter 7). We also intend to explore the use of other rule formats for algebraic properties in improving mechanized axiomatizations for bisimilarity. The ultimate goals of this research are to make automatically generated axiomatizations comparable to the known ones from the literature and to make the first steps towards the automatic generation of axiomatizations that are complete for open terms. The latter goal is a very ambitious one since obtaining complete axiomatizations of bisimilarity is a very hard research problem even for sufficiently rich fragments of specific process calculi; see, for instance, [11, 12, 22].

5.A Proving Theorem 5.3.10

Let R be the relation $\sim_{cc}$ restricted to closed terms. By definition, this relation contains all the desired pairs of terms of the form

$$(f(p_1,\ldots,p_j,\ldots,p_k,\ldots,p_n), f(p_1,\ldots,p_k,\ldots,p_j,\ldots,p_n)),$$

where $j,k \in K$ for some $K \in \prod_f$ and $j < k$. It therefore suffices to prove that R is a bisimulation relation. To this end, note, first of all, that R is symmetric since so is $\sim_{cc}$.

Consider now an arbitrary pair $(p,q) \in R$. Suppose that $p \xrightarrow{l} p'$ for some $l \in L$ and p'. We have to prove the existence of a q' such that $q \xrightarrow{l} q'$ and $(p',q') \in R$. This we show by induction on the definition of R, and proceed with a case analysis on

the reason why $(p, r) \in R$ holds. (Recall that R is the restriction of $\sim_{cc}$ to closed terms.)

1. If $p \equiv q$ then the claim holds since $\sim_{cc}$ is reflexive.

2. Assume that (p, q) is in R because $p = f(p_1, \ldots, p_n)$ and $q = f(q_1, \ldots, q_n)$, for some n-ary $f \in \Sigma$ and closed terms p_i and q_i such that $(p_i, q_i) \in R$ $(1 \leq i \leq n)$. As $p \xrightarrow{l} p'$, there are a rule

$$\textbf{(d)}\ \frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \xrightarrow{l_{ik}} \not\rightarrow \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{l} t}$$

   and a substitution σ such that

   - $\sigma(x_i) = p_i \xrightarrow{l_{ij}} \sigma(y_{ij})$ for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$,
   - $\sigma(x_i) = p_i \xrightarrow{l_{ik}} \not\rightarrow$ for all $1 \leq i \leq n$ and $1 \leq k \leq n_i$, and
   - $\sigma(t) = p'$.

   Let

$$\begin{aligned} X &\doteq \{x_i \mid 1 \leq i \leq n\} \text{ and} \\ Y &\doteq \{y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\}. \end{aligned}$$

   Our aim is to use the above rule to show that $q = f(q_1, \ldots, q_n) \xrightarrow{l} q'$ for some q' such that $(p', q') \in R$. To this end, we will now define a new substitution σ' in such a way that

   - σ' 'satisfies the premises of rule d',
   - $\sigma'(f(\vec{x})) = q$ and
   - $(\sigma(x), \sigma'(x)) \in R$, for each $x \in V$.

   We start with the following partial definition:

$$\sigma'(x) \doteq \begin{cases} q_i & \text{if } x = x_i \ (1 \leq i \leq n) \\ \sigma(x) & \text{if } x \in V \setminus (X \cup Y). \end{cases}$$

   Hitherto, we already have that $\sigma'(f(\vec{x})) = q$ and that $(\sigma(x), \sigma'(x)) \in R$ for each $x \in V \setminus Y$. Moreover, σ' satisfies the negative premises of d. Indeed, as $(p_i, q_i) \in R$ by assumption, the induction hypothesis yields that p_i and q_i

have the same set of initial actions. Therefore, since p_i satisfies the negative premises of d, so does q_i.

It remains to complete the definition of σ' for the variables in Y in such a way that σ' satisfies the positive premises of rule d and $(\sigma(x), \sigma'(x)) \in R$, for each $x \in Y$. To this end, consider a premise of d of the form $x_i \xrightarrow{l_{ij}} y_{ij}$, for some $1 \leq i \leq n$ and $1 \leq j \leq m_i$. Since $\sigma(x_i) = p_i$, $(p_i, q_i) \in R$ and $\sigma(x_i) \xrightarrow{l_{ij}} \sigma(y_{ij})$, it follows from the induction hypothesis that there exists some p'_{ij} such that $q_i \xrightarrow{l} p'_{ij}$ and $(\sigma(y_{ij}, p'_{ij}) \in R$. We let $\sigma'(y_{ij}) \doteq p'_{ij}$. Defining σ' for each $y_{ij} \in Y$ inductively, in this manner, concludes the proof since rule d instantiated with σ' gives us that $q \xrightarrow{l} \sigma'(t)$ and, by Lemma 5.3.6, $(\sigma(t), \sigma'(t)) \in R$. (Recall that R is just $\sim_{cc}$ restricted to closed terms.)

3. We are left to examine the case where

$$\begin{aligned} p &= f(p_1, \ldots, p_j, \ldots, p_k, \ldots, p_n) \text{ and} \\ q &= f(p_1, \ldots, p_k, \ldots, p_j, \ldots, p_n), \end{aligned}$$

for some $f \in \Sigma$ and $j, k \in K \in \prod_f$, with $j < k$. Since $p \xrightarrow{l} p'$, there are a rule

$$\textbf{(d)} \frac{\{x_i \xrightarrow{l_{i\ell}} y_{i\ell} \mid 1 \leq i \leq n, 1 \leq \ell \leq m_i\} \cup \{x_i \xrightarrow{l_{i\ell'}} \mid 1 \leq i \leq n, 1 \leq \ell' \leq n_i\}}{f(\vec{x}) \xrightarrow{l} t}$$

and a substitution σ such that

- $\sigma(x_i) = p_i \xrightarrow{l_{i\ell}} \sigma(y_{i\ell})$ for all $1 \leq i \leq n$ and $1 \leq \ell \leq m_i$,
- $\sigma(x_i) = p_i \xrightarrow{l_{i\ell'}}$ for all $1 \leq i \leq n$ and $1 \leq \ell' \leq n_i$, and
- $\sigma(t) = p'$.

As before, our aim is to show that $q = f(p_1, \ldots, p_k, \ldots, p_j, \ldots, p_n) \xrightarrow{l} q'$ for some q' such that $(p', q') \in R$. In what follows, for the sake of brevity, we refer to the set of premises of rule d as H. According to Definition 5.3.9, the transition system specification contains another rule of the following form:

$$\textbf{(d')} \frac{\{x'_i \xrightarrow{l'_{i\ell}} y'_{i\ell} \mid 1 \leq i \leq n, 1 \leq \ell \leq m'_i\} \cup \{x'_i \xrightarrow{l'_{i\ell'}} \mid 1 \leq i \leq n, 1 \leq \ell' \leq n'_i\}}{f(x'_1, \ldots, x'_n) \xrightarrow{l} t'}.$$

Moreover, there is a bijective mapping $\hbar$ over the set of variables such that

- $\hbar(x'_i) = x_i$ for $1 \leq i \leq n$ such that $i \neq j$ and $i \neq k$,
- $\hbar(x'_j) = x_k$ and $\hbar(x'_k) = x_j$,
- $\hbar(t') \sim_{cc} t$ and
- $\hbar(H') = H$, where H' stands for the collection of premises of d'.

Let $\sigma' = \sigma \circ \hbar$. It is not hard to see that σ' satisfies the premises of d'. Therefore,

$$\sigma'(f(x'_1, \ldots, x'_n)) = f(p_1, \ldots, p_k, \ldots, p_j, \ldots, p_n) = q \xrightarrow{l} \sigma'(t').$$

Since $\hbar(t') \sim_{cc} t$, by Lemma 5.3.6, we have that

$$\sigma'(t') = \sigma(\hbar(t')) \sim_{cc} \sigma(t).$$

Hence $(\sigma(t), \sigma'(t')) \in R$ and we are done.

4. Assume that $(p, q) \in R$ because, by shorter inferences, $(p, r) \in R$ and $(r, q) \in R$. By Lemma 5.3.5, r is closed. Suppose that $p \xrightarrow{l} p'$ for some closed term p'. By the inductive hypothesis, there is some closed term r' such that $r \xrightarrow{l} r'$ and $(p', r') \in R$. Using again the induction hypothesis, we have that $q \xrightarrow{l} q'$ and $(r', q') \in R$, for some q'. Since R is transitive, $(p', q') \in R$ and we are done.

This completes the proof of the theorem.

5.B Proof of Proposition 5.4.17

Consider an arbitrary disjoint extension of $\mathcal{T}'$. Assume that $f(p_1, \ldots, p_n) \xrightarrow{l} p$, for some closed terms $p_1, \ldots, p_n, p$. This is because there are a rule $d = \dfrac{H}{f(x_1, \ldots, x_n) \xrightarrow{l} t}$ and a closed substitution σ such that

- σ satisfies H,
- $\sigma(x_i) = p_i$, for each $i \in [n]$, and
- $\sigma(t) = p$.

Since the TSS under consideration is a disjoint extension of $\mathcal{T}'$, and thus of $\mathcal{T}$, there is some set of rules ρ_j, $j \in [\ell]$, and some rule $d' \in \rho_j$ such that $d \overset{K}{\smile} d'$. Letting,

without loss of generality, $d' = \dfrac{H'}{f(x_1,\ldots,x_n) \xrightarrow{l} t'}$, this means that there are some K-permutation π and some bijection $\hbar$ over variables such that

- $\hbar(x_i) = x_{\pi(i)}$, for each $i \in [n]$,
- $\hbar(t') \sim_{cc} t$, and
- $\hbar(H') = H$.

Consider now the closed substitution $\sigma \circ \hbar$. This substitution satisfies H' because σ satisfies H and $\hbar(H') = H$. Moreover,

$$(\sigma \circ \hbar)(f(x_1,\ldots,x_n)) = \sigma(f(x_{\pi(1)},\ldots,x_{\pi(n)})$$

and $(\sigma \circ \hbar)(t') \sim_{cc} \sigma(t) = p$, by Lemma 5.3.6. Furthermore, from the proof of Theorem 5.3.10 in Appendix 5.A, we have that $\sim_{cc}$ is a bisimulation, and therefore $(\sigma \circ \hbar)(t') \leftrightarroweq p$. Since there is a rule for f_j of the form $\dfrac{H'}{f_j(x_1,\ldots,x_n) \xrightarrow{l} t'}$, we have that

$$\sigma(f_j(x_{\pi(1)},\ldots,x_{\pi(n)})) \xrightarrow{l} (\sigma \circ \hbar)(t') \leftrightarroweq p.$$

Therefore it holds that

$$\sigma(\sum_{i=1}^{\ell} \sum \{f_i(x_{\pi(1)},\ldots,x_{\pi(n)}) \mid \pi \text{ is a } K\text{-permutation}\}) \xrightarrow{l} (\sigma \circ \hbar)(t') \leftrightarroweq p,$$

and we have found a matching transition, up to bisimilarity, from the instantiation of the right-hand side of equation (5.1) with σ.

We now check that each transition

$$t = \sum_{i=1}^{\ell} \sum \{f_i(p_{\pi(1)},\ldots,p_{\pi(n)}) \mid \pi \text{ a } K\text{-permutation}\} \xrightarrow{l} p$$

can be matched by $f(p_1,\ldots,p_n)$ up to bisimilarity. To this end, assume that, for some p, $t \xrightarrow{l} p$ This means that there are some $j \in [\ell]$ and some K-permutation π such that $f_j(p_{\pi(1)},\ldots,p_{\pi(n)}) \xrightarrow{l} p$. Since each rule for f_j is a rule for f, we have that $f(p_{\pi(1)},\ldots,p_{\pi(n)}) \xrightarrow{l} p$ also holds. As π is a K-permutation, $K \in \prod_f$ and $\mathcal{T}$ is in the comm-GSOS format with respect to $\prod^{\Sigma}$, by repeated use of Theorem 5.3.10, we have that $f(p_{\pi(1)},\ldots,p_{\pi(n)}) \leftrightarroweq f(p_1,\ldots,p_n)$. Therefore there is some p' such that $f(p_1,\ldots,p_n) \xrightarrow{l} p'$ and $p' \leftrightarroweq p$, which was to be shown.

Chapter 6

SOS Rule Formats for Idempotent Terms and Idempotent Unary Operators

6.1 Introduction

Over the last three decades, Structural Operational Semantics (SOS) [124] has proven to be a flexible and powerful way to specify the semantics of programming and specification languages. In this approach to semantics, the behaviour of syntactically correct language expressions is given in terms of a collection of state transitions that is specified by means of a set of syntax-driven inference rules. This behavioural description of the semantics of a language essentially tells one how the expressions in the language under definition behave when run on an idealized abstract machine.

Language designers often have expected algebraic properties of language constructs in mind when defining a language. For example, in the field of process algebras such as ACP [25], CCS [106] and CSP [95], operators such as nondeterministic and parallel composition are often meant to be commutative and associative with respect to bisimilarity [120]. Once the semantics of a language has been given in terms of state transitions, a natural question to ask is whether the intended algebraic properties do hold modulo the notion of behavioural semantics of interest. The typical approach to answer this question is to perform an *a posteriori verification*: based on the semantics in terms of state transitions, one proves the validity of the desired algebraic laws, which describe the expected semantic

properties of the various operators in the language. An alternative approach is to ensure the validity of algebraic properties *by design*, using the so called *SOS rule formats* [23]. In this approach, one gives *syntactic templates* for the inference rules used in defining the operational semantics for certain operators that guarantee the validity of the desired laws, thus obviating the need for an a posteriori verification. (See [5, 9, 10, 23, 61, 117] for examples of rule formats for algebraic properties in the literature on SOS.) The definition of SOS rule formats is based on finding a reasonably good trade-off between generality and ease of application. On the one hand, one strives to define a rule format that can capture as many examples from the literature as possible, including ones that may arise in the future. On the other, the rule format should be as easy to apply, and as syntactic, as possible.

The main advantage of the approach based on the development of rule formats is that one is able to verify the desired algebraic properties by syntactic checks that can be mechanized. Moreover, it is interesting to use rule formats for establishing semantic properties since the results so obtained apply to a broad class of languages. Last, but not least, these formats provide one with an understanding of the semantic nature of algebraic properties and of their connection with the syntax of SOS rules. This insight may serve as a guideline for language designers who want to ensure, a priori, that the constructs of a language under design enjoy certain basic algebraic properties.

Contribution The main aim of this chapter is to present a format of SOS rules that guarantees that some unary operation f is *idempotent* with respect to any notion of behavioural equivalence that includes bisimilarity. A unary operator f is idempotent if the equation $f(x) = f(f(x))$ holds. Examples of idempotent unary operators from the fields of language theory and process calculi are the unary Kleene star operator [96], the delay operator from SCCS [88, 105], the replication operator from the π-calculus [129] and the priority operator from [27].

It turns out that, in order to develop a rule format for unary idempotent operations that can deal with operations such as Kleene star and replication, one needs a companion rule format ensuring that terms of a certain form are idempotent for some binary operator. We recall that an element a of an algebra is said to be an *idempotent* for a binary operator $\odot$ if $a = a \odot a$. For example, the term x^*, where * denotes the Kleene star operation, is an idempotent for the sequential composition operation '$\cdot$' because the equation $x^* = x^* \cdot x^*$ holds. As a second contribution of

this chapter, we therefore offer an SOS rule format ensuring that certain terms are idempotent with respect to some binary operator. Both the rule formats we present in this chapter make an essential use of previously developed formats for algebraic properties such as associativity and commutativity [61, 117].

We provide a variety of examples showing that our rule formats can be used to establish the validity of several laws from the literature on process algebras dealing with idempotent unary operators and idempotent terms.

Structure The chapter is organized as follows. Section 6.2 reviews some standard definitions from the theory of SOS that will be used in the remainder of this study. We present our rule format for idempotent terms in Section 6.3. That rule format plays an important role in the definition of the rule format for idempotent unary operators that we give in Section 6.4. We discuss the results of the chapter and hint at directions for future work in Section 6.5.

This chapter is an extended version of [17]. Apart from offering the proofs of results that were announced without proof in that reference and a variety of examples of applications of our rule formats, this chapter also presents a generalization of the rule format for idempotent unary operations given in [17].

6.2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from process theory and the meta-theory of SOS that will be used in the remainder of the paper. We refer the interested reader to [14, 118] for further details.

Transition System Specifications in GSOS Format

Definition 6.2.1 (Signature, terms and substitutions)**.** *We let V denote an infinite set of variables with typical members $x, x', x_i, y, y', y_i, \ldots$. A* signature Σ *is a set of function symbols, each with a fixed arity. We call these symbols* operators *and usually denote them by $f, g, \ldots$. An operator with arity zero is called a* constant*. We define the set $\mathbb{T}(\Sigma)$ of* terms *over Σ (sometimes referred to as Σ-terms) as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

We use $t, t', t_i, u, \ldots$ *to range over terms. We write* $t_1 \equiv t_2$ *if* t_1 *and* t_2 *are syntactically equal. The function* $vars : \mathbb{T}(\Sigma) \to 2^V$ *gives the set of variables appearing in a term. The set* $\mathbb{C}(\Sigma) \subseteq \mathbb{T}(\Sigma)$ *is the set of* closed terms, *i.e., the set of all terms* t *such that* $vars(t) = \emptyset$. *We use* $p, p', p_i, q, \ldots$ *to range over closed terms. A* context *is a term with an occurrence of a hole* [ ] *in it.*

A substitution σ *is a function of type* $V \to \mathbb{T}(\Sigma)$. *We extend the domain of substitutions to terms homomorphically. If the range of a substitution is included in* $\mathbb{C}(\Sigma)$, *we say that it is a* closed substitution. *For a substitution* σ *and sequences* $x_1, \ldots, x_n$ *and* $t_1, \ldots, t_n$ *of distinct variables and of terms, respectively, we write* $\sigma[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ *for the substitution that maps each variable* x_i *to* t_i $(1 \leq i \leq n)$ *and agrees with* σ *on all of the other variables. When* σ *is the identity function over variables, we abbreviate* $\sigma[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ *to* $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$.

The GSOS format is a widely studied format of deduction rules in transition system specifications proposed by Bloom, Istrail and Meyer [46, 49]. Transition system specifications whose rules are in the GSOS format enjoy many desirable properties, and several studies in the literature on the meta-theory of SOS have focused on them—see, for instance, [3, 2, 6, 14, 20, 35]. In this study we shall also focus on transition system specifications in the GSOS format, which we now proceed to define.

Definition 6.2.2 (GSOS Format [49])**.** *A deduction rule for an operator* f *of arity* n *is in the* GSOS format *if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \xnrightarrow{l_{ik}} \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{l} C[\vec{x}, \vec{y}]} \tag{6.1}$$

where the x_i*'s and the* y_{ij}*'s* $(1 \leq i \leq n$ *and* $1 \leq j \leq m_i)$ *are all distinct variables,* m_i *and* n_i *are natural numbers,* $C[\vec{x}, \vec{y}]$ *is a* Σ*-term with variables including at most the variables* x_i*'s and* y_{ij}*'s, and the* l_{ij}*'s and* l *are action labels (or simply actions). The above rule is said to be* f-defining *and* l-emitting.

A transition system specification (TSS) in the GSOS format $\mathcal{T}$ *is a triple* (Σ, L, D) *where* Σ *is a finite signature,* L *is a finite set of labels, and* D *is a finite set of deduction rules in the GSOS format. The collection of* f*-defining and* l*-emitting rules in a set* D *of GSOS rules is denoted by* $D(f, l)$.

Example 6.2.3. *An example of a TSS in the GSOS format is the one describing the semantics of BCCSP [81]. The signature for this TSS contains the operators* **0** *(of arity*

zero), $a.\_$ ($a \in L$) and $\_+\_$. The standard deduction rules for these operators are listed below, where a ranges over L.

$$\frac{}{a.x_1 \xrightarrow{a} x_1} \qquad \frac{x_1 \xrightarrow{a} x_1'}{x_1 + x_2 \xrightarrow{a} x_1'} \qquad \frac{x_2 \xrightarrow{a} x_2'}{x_1 + x_2 \xrightarrow{a} x_1'}$$

Informally, the intent of a GSOS rule of the form (6.1) is as follows. Suppose that we are wondering whether $f(\vec{p})$ is capable of taking an l-step. We look at each f-defining and l-emitting rule in turn. We inspect each positive premise $x_i \xrightarrow{l_{ij}} y_{ij}$, checking if p_i is capable of taking an l_{ij}-step for each j and if so calling the l_{ij}-children q_{ij}. We also check the negative premises: if p_i is *in*capable of taking an l_{ik}-step for each k. If so, then the rule *fires* and $f(\vec{p}) \xrightarrow{l} C[\vec{p}, \vec{q}]$. This means that the transition relation $\rightarrow$ associated with a TSS in the GSOS format is the one defined by the rules using structural induction over closed Σ-terms. This transition relation is the unique sound and supported transition relation. Here *sound* means that whenever a closed substitution σ 'satisfies' the premises of a rule of the form (6.1), then $\sigma(f(\vec{x})) \xrightarrow{l} \sigma(C[\vec{x}, \vec{y}])$. On the other hand, *supported* means that any transition $p \xrightarrow{l} q$ can be obtained by instantiating the conclusion of a rule of the form (6.1) with a substitution that satisfies its premises. We refer the interested reader to [46, 49] for the precise definition of $\rightarrow$ and much more information on GSOS languages. The above informal description of the transition relation associated with a TSS in GSOS format suffices to follow the technical developments in the remainder of the paper.

Remark 6.2.4. *In this paper, we restrict ourselves to TSSs in GSOS format for the sake of simplicity, though the results also hold for tyft/tyxt. The rule formats we present in what follows can be extended to arbitrary TSSs at the price of considering the so-called* three-valued stable models. *See [14] for a survey introduction to three-valued stable models.*

Bisimilarity Terms built using operators from the signature of a TSS are usually considered modulo some notion of behavioural equivalence, which is used to indicate when two terms describe 'essentially the same behaviour'. The notion of behavioural equivalence that we will use in this paper is the following, classic notion of bisimilarity [106, 120].

Definition 6.2.5 (Bisimilarity)**.** *Let $\mathcal{T}$ be a TSS in GSOS format with signature Σ. A relation $R \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* bisimulation *if and only if R is symmetric and, for all*

$p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ *and* $l \in L$,

$$(p_0 \; R \; p_1 \wedge p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\, (p_1 \xrightarrow{l} p_1' \wedge p_0' \; R \; p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ *are called* bisimilar, *denoted by* $\mathcal{T} \vdash p_0 \leftrightarroweq p_1$ *(or simply by* $p_0 \leftrightarroweq p_1$ *when* $\mathcal{T}$ *is clear from the context), when there exists a bisimulation* R *such that* $p_0 \; R \; p_1$. *We refer to the relation* $\leftrightarroweq$ *as* bisimilarity.

It is well known that $\leftrightarroweq$ is an equivalence relation over $\mathbb{C}(\Sigma)$. Any equivalence relation $\sim$ over closed terms in a TSS $\mathcal{T}$ is extended to open terms in the standard fashion, i.e., for all $t_0, t_1 \in \mathbb{T}(\Sigma)$, the equation $t_0 = t_1$ holds over $\mathcal{T}$ modulo $\sim$ (sometimes abbreviated to $t_0 \sim t_1$) if, and only if, $\mathcal{T} \vdash \sigma(t_0) \sim \sigma(t_1)$ for each closed substitution σ.

Definition 6.2.6. *Let* Σ *be a signature. An equivalence relation* $\sim$ *over* Σ*-terms is a* congruence *if, for all* $f \in \Sigma$ *and closed terms* $p_1, \ldots, p_n, q_1, \ldots, q_n$, *where* n *is the arity of* f, *if* $p_i \sim q_i$ *for each* $i \in \{1, \ldots, n\}$ *then* $f(p_1, \ldots, p_n) \sim f(q_1, \ldots, q_n)$.

Remark 6.2.7. *Let* Σ *be a signature and let* $\sim$ *be a congruence. It is easy to see that, for all* $f \in \Sigma$ *and terms* $t_1, \ldots, t_n, u_1, \ldots, u_n$, *where* n *is the arity of* f, *if* $t_i \sim u_i$ *for each* $i \in \{1, \ldots, n\}$ *then* $f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$.

The following result is well known [46].

Proposition 6.2.8. $\leftrightarroweq$ *is a congruence for any TSS in GSOS format.*

The above proposition is a typical example of a result in the meta-theory of SOS: it states that if the rules in a TSS satisfy some syntactic constraint, then some semantic result is guaranteed to hold. In the remainder of this paper, following the work presented in, e.g., [5, 9, 10, 23, 61, 117], we shall present a rule format ensuring that certain unary operations are idempotent. This rule format will rely on one yielding that terms of a certain form are idempotent for some binary operator. For this reason, we present first the latter rule format in the subsequent section.

6.3 A rule format for idempotent terms

Definition 6.3.1 (Idempotent term)**.** *Let* Σ *be a signature. Let* f *and* $\odot$ *be, respectively, a unary and a binary operator in* Σ. *We say that* $f(x)$ *is an* idempotent term *for* $\odot$ *with*

respect to an equivalence relation $\sim$ *over* $\mathbb{T}(\Sigma)$ *if the following equation holds:*

$$f(x) \sim f(x) \odot f(x). \tag{6.2}$$

In what follows, we shall present some syntactic requirements on the SOS rules defining the operators f and $\odot$ that guarantee the validity of equation (6.2) with respect to bisimilarity, and therefore any notion of equivalence that is coarser than it. In order to motivate the syntactic constraints of the rule format, let us consider the unary replication operator '!', which is familiar from the theory of the π-calculus (see, e.g., [129]), and the binary interleaving parallel composition '$\|$', which appears in, amongst others, ACP [44], CCS [104], and CSP [95, 126]. The rules for these operators are given below, where a ranges over the set of action labels L.

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \,\|\, !x} \qquad \frac{x \xrightarrow{a} x'}{x \,\|\, y \xrightarrow{a} x' \,\|\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,\|\, y \xrightarrow{a} x \,\|\, y'} \tag{6.3}$$

It is well known that $!x$ is an idempotent term for $\|$ modulo any notion of equivalence that includes bisimilarity. Indeed, the equation

$$!x = (!x) \,\|\, (!x)$$

is one of the laws for the structural congruence over the π-calculus with replication considered in, e.g., [66].

It is instructive to try and find out why the above law holds by considering the interplay between the transition rules for '!' and those for '$\|$', rather than considering the transitions that are possible for all the closed instantiations of the terms $!x$ and $(!x) \,\|\, (!x)$. To this end, consider the rule for replication for some action a. The effect of this rule can be mimicked by the term $(!x) \,\|\, (!x)$ by means of a combination of the instance of the first rule for $\|$ in (6.3) for action a and of the rule for replication. When we do so, the appropriate instantiation of the target of the conclusion of the first rule for $\|$ is the term

$$(x' \,\|\, y)[x' \mapsto x' \,\|\, !x,\ y \mapsto !x] = (x' \,\|\, !x) \,\|\, !x.$$

Note that the target of the conclusion of the rule for replication, namely $x' \,\|\, !x$, and the above term can be proved equal using associativity of $\|$, which is well known, and the version of axiom (6.2) for replication and parallel composition, as

follows:

$$(x' \parallel !x) \parallel !x = x' \parallel (!x \parallel !x) = x' \parallel !x.$$

The validity of the associativity law for $\parallel$ is guaranteed by the rule format for associativity given in [61, Definition 8]. On the other hand, as shown in the remainder of the section, the soundness of the use of equation (6.2) can be justified using coinduction [128].

Consider instead the combination of the instance of the second rule for $\parallel$ in (6.3) for action a and of the rule for replication for that action. When we do so, the appropriate instantiation of the target of the conclusion of the second rule for $\parallel$ is the term

$$(x \parallel y')[x \mapsto !x,\ y \mapsto x' \parallel !x] = !x \parallel (x' \parallel !x).$$

Note that the target of the conclusion of the rule for replication, namely $x' \parallel !x$, and the above term can be proved equal using commutativity and associativity of $\parallel$, which are well known, and the version of axiom (6.2) for replication and parallel composition, as follows:

$$!x \parallel (x' \parallel !x) = (x' \parallel !x) \parallel !x = x' \parallel (!x \parallel !x) = x' \parallel !x.$$

The validity of the commutativity law for $\parallel$ is guaranteed by the rule format for commutativity given in [117].

The above discussion hints at the possibility of defining an SOS rule format guaranteeing the validity of equation (6.2) building on SOS rule formats for algebraic properties like associativity and commutativity of operators [23], and on a coinductive use of equation (6.2) itself. The technical developments to follow will offer a formalization of this observation.

Our definition of the rule format is based on a syntactically defined equivalence relation over terms that is sufficient to handle the examples from the literature we have met so far.

Definition 6.3.2 (The relation $\leftrightsquigarrow$)**.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in GSOS format.*

1. *The relation $\leftrightsquigarrow$ is the least equivalence relation over $\mathbb{T}(\Sigma)$ that satisfies the following clauses:*

   - *$f(t,u) \leftrightsquigarrow f(u,t)$, if f is a binary operator in Σ and the commutativity rule format from [117] applies to f,*

- $f(t, f(t', u)) \leftrightsquigarrow f(f(t, t'), u)$, *if* f *is a binary operator in* Σ *and one of the associativity rule formats from [61] applies to* f, *and*
- $C[t] \leftrightsquigarrow C[t']$, *if* $t \leftrightsquigarrow t'$, *for each context* $C[\ ]$.

2. *Let* f *and* $\odot$ *be, respectively, a unary and a binary operator in* Σ. *We write* $t \downarrow_{f,\odot} u$ *if, and only if, there are some* t' *and* u' *such that* $t \leftrightsquigarrow t'$, $u \leftrightsquigarrow u'$, *and* $t' = u'$ *can be proved by possibly using one application of an instantiation of axiom (6.2) in a context—that is, either* $t' \equiv u'$, *or* $t' = C[f(t'')]$ *and* $u' = C[f(t'') \odot f(t'')]$, *for some context* $C[\ ]$ *and term* t'', *or vice versa.*

Example 6.3.3. *Consider the terms* $!x \parallel (x' \parallel !x)$ *and* $x' \parallel !x$. *Then*

$$x' \parallel !x \downarrow_{!,\parallel} !x \parallel (x' \parallel !x).$$

Indeed, $!x \parallel (x' \parallel !x) \leftrightsquigarrow x' \parallel (!x \parallel !x)$, *because the rules for* $\parallel$ *are in the associativity and commutativity rule formats from [61, 117], and* $x' \parallel !x = x' \parallel (!x \parallel !x)$ *can be proved using one application of the relevant instance of axiom (6.2) in the context* $x' \parallel [\ ]$.

Remark 6.3.4. *The definition of the relation* $\leftrightsquigarrow$ *can be easily strengthened by adding more clauses, provided their soundness with respect to bisimilarity can be 'justified syntactically'. Moreover, in the definition of the relation* $\downarrow_{f,\odot}$, *we could allow for any number of applications of axiom (6.2) in context. The current definition suffices to handle all the examples from the literature we have met so far.*

Lemma 6.3.5. *Let* $\mathcal{T} = (\Sigma, L, D)$ *be a TSS in the GSOS format, and let* $t, t' \in \mathbb{T}(\Sigma)$. *If* $t \leftrightsquigarrow t'$ *then*

1. $t \leftrightarrows t'$,
2. $vars(t) = vars(t')$ *and*
3. $\sigma(t) \leftrightsquigarrow \sigma(t')$, *for each substitution* σ.

Proof. All the claims can be shown by induction on the definition of $\leftrightsquigarrow$. The soundness modulo $\leftrightarrows$ of the rewrite rules in Definition 6.3.2(1) is guaranteed by results in [61, 117] and by Proposition 6.2.8. □

We are now ready to present an SOS rule format guaranteeing the validity of equation (6.2). The aim of the rule format is to ensure that the following properties hold for each closed term p, whenever the rules for a unary operator f and a binary operator $\odot$ satisfy the given constraints:

- if $f(p) \xrightarrow{a} p'$ for some p', then there is some q' such that $f(p) \odot f(p) \xrightarrow{a} q'$ and the bisimilarity between p' and q' can be justified using the relation $\downarrow_{f,\odot}$;

- if $f(p) \odot f(p) \xrightarrow{a} q'$ for some q', then there is some p' such that $f(p) \xrightarrow{a} p'$ and the bisimilarity between p' and q' can be justified using the relation $\downarrow_{f,\odot}$.

Intuitively, condition 1 in the definition below ensures that the former property holds by requiring that each rule for f can be used to derive suitable matching transitions from terms of the form $f(p) \odot f(p)$. On the other hand, conditions 2 and 3 guarantee that the latter property holds by requiring that any applicable combination of rules for f and $\odot$ can be 'simulated' by a rule for f.

Definition 6.3.6 (Rule format for idempotent terms)**.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let f and $\odot$ be, respectively, a unary and a binary operator in Σ. We say that the rules for f and $\odot$ in $\mathcal{T}$ are in the* rule format for idempotent terms *if either the rules for $\odot$ are in the rule format for idempotence from [5] or the following conditions are met:*

1. *For each f-defining rule in D, say*

$$\frac{H}{f(x) \xrightarrow{a} t},$$

there is some $\odot$-defining rule

$$\frac{H'}{x_1 \odot x_2 \xrightarrow{a} u},$$

such that

   (a) $H' \subseteq \{x_1 \xrightarrow{a} y_1,\ x_2 \xrightarrow{a} y_2\}$, *and*

   (b) $t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]$.

2. *Each $\odot$-defining rule has the form*

$$\frac{\{x_i \xrightarrow{a} y_i\} \cup \{x_1 \xrightarrow{a_j} y_j \mid j \in J\} \cup \{x_2 \xrightarrow{b_k} z_k \mid k \in K\}}{x_1 \odot x_2 \xrightarrow{a} u}, \tag{6.4}$$

   where $i \in \{1, 2\}$, J and K are index sets, and $vars(u) \subseteq \{x_1, x_2, y_i\}$.

3. *Let r be an $\odot$-defining rule of the form (6.4) such that $D(f, a_j) \neq \emptyset$, for each $j \in J$, and $D(f, b_k) \neq \emptyset$, for each $k \in K$. Let*

$$\frac{H}{f(x) \xrightarrow{a} t}$$

be a rule for f. Then

$$t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_i \mapsto t].$$

Theorem 6.3.7. *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let f and $\odot$ be, respectively, a unary and a binary operator in Σ. Assume that the rules for f and $\odot$ in $\mathcal{T}$ are in the rule format for idempotent terms. Then equation (6.2) holds over $\mathcal{T}$ modulo bisimilarity.*

The proof of this result may be found in Appendix 6.A.

Example 6.3.8 (Replication and parallel composition)**.** *The unary replication operator and the parallel composition operator, whose rules we presented in (6.3), are in the rule format for idempotent terms. Indeed, we essentially carried out the verification of the conditions in Definition 6.3.6 when motivating the constraints of the rule format and the relation $\leftrightsquigarrow$. Therefore, Theorem 6.3.7 yields the soundness, modulo bisimilarity, of the well-known equation*

$$!x = (!x) \parallel (!x).$$

Example 6.3.9 (Kleene star and sequential composition)**.** *Assume that the set of action labels L contains a distinguished action $\checkmark$, which is used to signal the successful termination of a process.*

Consider the unary Kleene star operator '$\_^*$' [96] and the binary sequential composition operator '$\cdot$' given by the rules

$$\frac{}{x^* \xrightarrow{\checkmark} \delta} \qquad \frac{x \xrightarrow{a} x'}{x^* \xrightarrow{a} x' \cdot (x^*)} \ (a \neq \checkmark)$$

$$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \ (a \neq \checkmark) \qquad \frac{x \xrightarrow{\checkmark} x',\ y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'} \ (a \in L),$$

where δ is a constant. These rules satisfy the requirements of the rule format for idempotent terms. Indeed, to verify condition 1 in Definition 6.3.6, observe that

- *the first rule for the Kleene star operator can be 'matched' by the instance of the second rule schema for $\cdot$ with $a = \checkmark$, and*
- *the second rule for the Kleene star operator can be 'matched' by the equally-labelled instance of the second rule schema for $\cdot$.*

Condition 2 is easily seen to hold. To check condition 3, we examine all pairs of equally-labelled rules for $\cdot$ and the Kleene star operator. By way of example, let us look at the pair

consisting of the rules

$$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}\ (a \neq \checkmark) \qquad \frac{x \xrightarrow{a} x'}{x^* \xrightarrow{a} x' \cdot (x^*)}\ (a \neq \checkmark).$$

Then

$$\begin{aligned}(x' \cdot y)[x \mapsto x^*, y \mapsto x^*, x' \mapsto x' \cdot (x^*)] &= (x' \cdot (x^*)) \cdot x^* \\ &\leftrightsquigarrow\ x' \cdot (x^* \cdot x^*) \\ &= x' \cdot (x^*) \qquad \text{by (6.2).}\end{aligned}$$

The second step in the above argument is justified since the associativity rule format from [61, Definition 10] applies to $\cdot$. Therefore, Theorem 6.3.7 yields the soundness, modulo bisimilarity, of the well-known equation

$$x^* = x^* \cdot x^*.$$

Example 6.3.10 (Perpetual loop and sequential composition)**.** *Consider the unary perpetual loop operator '$\_^\omega$' from [69] given by the rules*

$$\frac{x \xrightarrow{a} x'}{x^\omega \xrightarrow{a} x' \cdot (x^\omega)}\ (a \neq \checkmark)$$

and the binary sequential composition operator '$\cdot$' given by the rules in Example 6.3.9. These rules satisfy the requirements of the rule format for idempotent terms. The conditions in Definition 6.3.6 can be checked along the lines of Example 6.3.9. Note that, since there is no $\checkmark$-emitting rule for the perpetual loop operator, the premises of rules of the form

$$\frac{x \xrightarrow{\checkmark} x',\ y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}\ (a \in L)$$

cannot be all met when x is instantiated with x^ω and therefore condition 3 in Definition 6.3.6 holds vacuously for those rules. Thus, Theorem 6.3.7 yields the soundness, modulo bisimilarity, of the well-known equation

$$x^\omega = x^\omega \cdot x^\omega.$$

6.4 A rule format for idempotent unary operators

Definition 6.4.1 (Idempotent unary operator)**.** *Let Σ be a signature. Let f be a unary operator in Σ. We say that $f(x)$ is* idempotent *with respect to an equivalence relation $\sim$ over $\mathbb{T}(\Sigma)$ if the following equation holds:*

$$f(x) \sim f(f(x)). \tag{6.5}$$

Example 6.4.2 (Delay operator)**.** *The rules for the unary delay operator δ from SCCS [88, 105] are as follows, where 1 is a distinguished action symbol in L that is used to denote a delay of one time unit:*

$$\frac{}{\delta x \xrightarrow{1} \delta x} \qquad \frac{x \xrightarrow{a} x'}{\delta x \xrightarrow{a} x'} \ (a \in L).$$

It is well known that δ is idempotent modulo bisimilarity.

Example 6.4.3 (Prefix iteration)**.** *The delay operator we presented in Example 6.4.2 is a special case of the unary prefix iteration operator $a^*\_$ from [68]. The rules for this operator are as follows:*

$$\frac{}{a^*x \xrightarrow{a} a^*x} \qquad \frac{x \xrightarrow{b} x'}{a^*x \xrightarrow{b} x'} \ (b \in L).$$

It is well known that $a^*\_$ is idempotent modulo bisimilarity.

Example 6.4.4 (Hiding operator)**.** *Assume that τ is a distinguished action in L that is used to label internal transitions of a system. For each $I \subseteq L \setminus \{\tau\}$, the rules for the unary hiding operator τ_I familiar from ACP [40] and CSP [95] are as follows:*

$$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')} \ (a \in L) \qquad \frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')} \ (a \notin L).$$

It is well known that τ_I is idempotent modulo bisimilarity.

In what follows, we shall present some syntactic requirements on the SOS rules defining a unary operator f that guarantee the validity of equation (6.5). The rule format for idempotent unary operators will rely on the one for idempotent terms given in Definition 6.3.6.

In order to motivate the use of the rule format for idempotent terms in the definition of the one for idempotent unary operators, consider the replication operator

whose rules were introduced in (6.3). As is well known, the equation

$$!x = !(!x)$$

holds modulo bisimilarity. The validity of this equation can be 'justified' using the transition rules for '!' as follows. Consider the rule for replication for some action a, namely

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \parallel !x}.$$

The effect of this rule can be mimicked by the term $!(!x)$ by using the same rule twice. When we do so, the appropriate instantiation of the target of the conclusion of the rule for '!' is the term

$$(x' \parallel !x)[x' \mapsto x' \parallel !x,\ x \mapsto !x] = (x' \parallel !x) \parallel !(!x).$$

Note that the target of the conclusion of the rule for replication, namely $x' \parallel !x$, and the above term can be proved equal using associativity of $\parallel$, the version of axiom (6.2) for replication and parallel composition, and the version of axiom (6.5) for replication, as follows:

$$(x' \parallel !x) \parallel !(!x) = (x' \parallel !x) \parallel !x = x' \parallel (!x \parallel !x) = x' \parallel !x.$$

As mentioned in Example 6.3.8, the validity of the version of axiom (6.2) for replication and parallel composition is guaranteed by Theorem 6.3.7. On the other hand, as shown in the remainder of the section, the soundness of the use of equation (6.5) can be justified using coinduction [128].

As we did in the definition of the rule format for idempotent terms presented in Definition 6.3.6, in stating the requirements of the rule format for idempotent unary operators, we shall employ a syntactically defined equivalence relation over terms that is sufficient to handle the examples from the literature we have met so far.

Definition 6.4.5 (The relation $\leftrightarrow$)**.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format.*

1. *The relation $\leftrightarrow$ is the least equivalence relation over $\mathbb{T}(\Sigma)$ that satisfies the following clauses:*

   - *$f(t) \leftrightarrow f(t) \odot f(t)$, if the rules for f and $\odot$ in $\mathcal{T}$ are in the rule format for idempotent terms from Definition 6.3.6,*

- $f(t,u) \leftrightarrow f(u,t)$, *if* f *is a binary operator in* Σ *and the commutativity rule format from [117] applies to* f,
- $f(t,f(t',u)) \leftrightarrow f(f(t,t'),u)$, *if* f *is a binary operator in* Σ *and one of the associativity rule formats from [61] applies to* f, *and*
- $C[t] \leftrightarrow C[t']$, *if* $t \leftrightarrow t'$, *for each context* $C[\ ]$.

2. *Let* f *be a unary operator in* Σ. *We write* $t \Downarrow_f u$ *if, and only if, there are some* t' *and* u' *such that* $t \leftrightarrow t'$, $u \leftrightarrow u'$, *and* $t' = u'$ *can be proved by possibly using one application of an instantiation of axiom (6.5) in a context—that is, either* $t' \equiv u'$, *or* $t' = C[f(t'')]$ *and* $u' = C[f(f(t''))]$, *for some context* $C[\ ]$ *and term* t'', *or vice versa.*

Example 6.4.6. *Consider the terms* $(x' \parallel !x) \parallel !(!x)$ *and* $x' \parallel !x$. *Then*

$$x' \parallel !x \Downarrow_! (x' \parallel !x) \parallel !(!x).$$

Indeed, $x' \parallel !x \leftrightarrow (x' \parallel !x) \parallel !x$, *using the relevant instance of axiom (6.2) and associativity of* $\parallel$, *and* $(x' \parallel !x) \parallel !x = (x' \parallel !x) \parallel !(!x)$ *can be proved using one application of the relevant instance of axiom (6.5) in the context* $(x' \parallel !x) \parallel [\ ]$.

Lemma 6.4.7. *Let* $\mathcal{T} = (\Sigma, L, D)$ *be a TSS in the GSOS format, and let* $t, t' \in \mathbb{T}(\Sigma)$. *If* $t \leftrightarrow t'$ *then*

1. $t \leftrightarrows t'$,
2. $vars(t) = vars(t')$ *and*
3. $\sigma(t) \leftrightarrow \sigma(t')$, *for each substitution* σ.

Proof. The lemma can be shown along the lines of the proof of Lemma 6.3.5, using Theorem 6.3.7 to justify the soundness modulo bisimilarity of applications of the clause $f(t) \leftrightarrow f(t) \odot f(t)$, when the rules for f and $\odot$ in $\mathcal{T}$ are in the rule format for idempotent terms from Definition 6.3.6. □

As indicated by the SOS rules for the hiding operator presented in Example 6.4.4, a rule format for idempotent unary operators that can deal with that operator should allow for some 'renaming' of transition labels in rules. Indeed, the operator τ_I relabels a actions of its argument term into τ for each $a \in I$. However, the renamings $h : L \to L$ that should be allowed by a rule format for idempotent unary operators are, not surprisingly, limited to the *idempotent* ones—that is, to those satisfying $h^2 = h$. It is not hard to see that a renaming $h : L \to L$ is idempotent if, and only if, $h(a) = a$ for each a contained in the range of h. Conditions 3 and 4

in the following definition formalize this observation and, in conjunction with the other requirements in that definition, guarantee that each transition from a closed term of the form $f(p)$ can be 'matched' by a transition from $f(f(p))$, and vice versa.

Definition 6.4.8 (Rule format for idempotent unary operators)**.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let f be a unary operator in Σ. We say that the rules for f are in the rule format for* idempotent unary operators *if the following conditions are met:*

1. *Each rule for f in D has the form*

$$\frac{H \cup \{x \overset{b_j}{\nrightarrow} \mid j \in J\}}{f(x) \overset{a}{\to} t}, \tag{6.6}$$

   where either

   (a) *$H \subseteq \{x \overset{a}{\to} x'\}$ and*

   (b) *$H = \{x \overset{a}{\to} x'\}$ if J is non-empty.*

   or $J = \emptyset$ and $H = \{x \overset{b}{\to} x'\}$ for some $b \neq a$. In the latter case, we call (6.6) a renaming rule *and say that b* is renamed by rule (6.6).

2. *If some rule for f of the form (6.6) has a premise of the form $x \overset{b}{\nrightarrow}$, then each b-emitting and f-defining rule has a positive premise of the form $x \overset{b}{\to} x'$.*

3. *If some a-emitting and f-defining rule is a renaming rule, then there is an f-defining rule of the form*

$$\frac{x \overset{a}{\to} x'}{f(x) \overset{a}{\to} f(x')}. \tag{6.7}$$

4. *If b is renamed by some rule of the form (6.6) then each b-emitting and f-defining rule has the form*

$$\frac{x \overset{b}{\to} x'}{f(x) \overset{b}{\to} t'}, \tag{6.8}$$

   for some term t'.

5. *Consider a rule for f of the form (6.6). Then one of the following conditions is met:*

   (a) *H is empty and $t \Downarrow_f t[x \mapsto f(x)]$,*

(b) $H = \{x \xrightarrow{a} x'\}$ *and, for each a-emitting rule for* f

$$\frac{H'}{f(x) \xrightarrow{a} t'},$$

we have that

$$t' \Downarrow_f t[x \mapsto f(x), x' \mapsto t'],$$

or

(c) $H = \{x \xrightarrow{b} x'\}$ *with* $a \neq b$ *and, for each b-emitting rule for* f *of the form (6.8),*

$$t \Downarrow_f t[x \mapsto f(x), x' \mapsto t'].$$

Theorem 6.4.9. *Let* $\mathcal{T} = (\Sigma, L, D)$ *be a TSS in the GSOS format. Let* f *be a unary operator in* Σ*. Assume that the rules for* f *in* $\mathcal{T}$ *are in the rule format for idempotent unary operators. Then equation (6.5) holds over* $\mathcal{T}$ *modulo bisimilarity.*

The proof of this result may be found in Appendix 6.B.

Remark 6.4.10. *Condition 1b in Definition 6.4.8 requires that, in rules of the form (6.6),* $H = \{x \xrightarrow{a} x'\}$ *if* J *is non-empty. This requirement is necessary for the validity of Theorem 6.4.9. To see this, consider the unary operator* f *with rules*

$$\frac{x \overset{b}{\nrightarrow}}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{x \xrightarrow{b} x',\ x \overset{c}{\nrightarrow}}{f(x) \xrightarrow{b} \mathbf{0}} \qquad \frac{x \xrightarrow{c} x',\ x \overset{b}{\nrightarrow}}{f(x) \xrightarrow{c} \mathbf{0}}.$$

The rules for f *satisfy all the conditions in Definition 6.4.8 apart from the requirement in condition 1b that all rules have positive premises when they have negative ones.*

It is not hard to see that $f(b.\mathbf{0} + c.\mathbf{0})$ *has no outgoing transitions. On the other hand, using the a-emitting rule for* f*, we have that*

$$f(f(b.\mathbf{0} + c.\mathbf{0})) \xrightarrow{a} \mathbf{0}.$$

Therefore, $f(b.\mathbf{0} + c.\mathbf{0}) \not\leftrightarrows f(f(b.\mathbf{0} + c.\mathbf{0}))$*, and the equation* $f(x) \leftrightarrows f(f(x))$ *does not hold.*

Remark 6.4.11. *Condition 2 in Definition 6.4.8 is necessary for the validity of Theorem 6.4.9. To see this, consider the unary operator* f *with rules*

$$\frac{x \overset{b}{\nrightarrow},\ x \xrightarrow{a} x'}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{}{f(x) \xrightarrow{b} \mathbf{0}}.$$

The rules for f satisfy all the conditions in Definition 6.4.8 apart from the requirement in condition 2. Observe that $f(a.\mathbf{0}) \xrightarrow{a} \mathbf{0}$. On the other hand, $f(f(a.\mathbf{0})) \not\xrightarrow{a}$. Therefore, $f(a.\mathbf{0}) \not\leftrightarrows f(f(a.\mathbf{0}))$, and the equation $f(x) \leftrightarrows f(f(x))$ does not hold.

6.4.1 Examples

We now present some examples of applications of the rule format given in Definition 6.4.8.

Example 6.4.12 (Delay operator)**.** *Consider the delay operator δ introduced in Example 6.4.2. Each rule for δ is of the form (6.6), meeting condition 1 in Definition 6.4.8. To see that condition 5 is also met, observe that*

- $\delta x \Downarrow_f \delta(\delta x) = (\delta x)[x \mapsto \delta x, x' \mapsto \delta x]$,
- $x' \Downarrow_f x' = x'[x \mapsto \delta x, x' \mapsto x']$, *and*
- $\delta x \Downarrow_f x'[x \mapsto \delta x, x' \mapsto \delta x]$.

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$\delta x = \delta(\delta x).$$

The prefix iteration operator discussed in Example 6.4.3 is handled in similar fashion.

Example 6.4.13 (Encapsulation)**.** *Consider the classic unary encapsulation operators ∂_H from ACP [25], where $H \subseteq L$, with rules*

$$\frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \quad a \notin H.$$

It is a simple matter to check that the above rules meet all the conditions in Definition 6.4.8. In particular,

$$\partial_H(x') \Downarrow_{\partial_H} \partial_H(\partial_H(x')) = \partial_H(x')[x \mapsto \partial_H(x), x' \mapsto \partial_H(x')].$$

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$\partial_H(x) = \partial_H(\partial_H(x)).$$

Example 6.4.14 (Priority). *Assume that $<$ is an irreflexive partial ordering over L. The priority operator θ from [27] has rules*

$$\frac{x \xrightarrow{a} x',\ (x \overset{b}{\nrightarrow} \text{ for each } b \text{ such that } a < b)}{\theta(x) \xrightarrow{a} \theta(x')} \qquad (a \in L).$$

It is not hard to see that the rules for θ satisfy the conditions in Definition 6.4.8. In particular, for each $a \in L$, the only a-emitting rule for θ has a positive premise of the form $x \xrightarrow{a} x'$. Hence, condition 1b in Definition 6.4.8 is met.

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$\theta(x) = \theta(\theta(x)).$$

Example 6.4.15 (Replication). *Consider the replication operator '$!$' whose rules were given in (6.3). We claim that the rules for '$!$' satisfy the conditions in Definition 6.4.8. Indeed, the rules for '$!$' are of the form (6.6) and, as we observed earlier in Example 6.4.6,*

$$x' \parallel !x \Downarrow_! (x' \parallel !x) \parallel !(!x) = (x' \parallel !x)[x \mapsto !x, x' \mapsto x' \parallel !x].$$

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$!x = !(!x).$$

Example 6.4.16 (Kleene star). *Consider the unary Kleene star operator '$\_^*$' whose rules were given in Example 6.3.9. We claim that the rules for '$\_^*$' satisfy the conditions in Definition 6.4.8. Indeed, observe, first of all, that the rules for '$\_^*$' are of the form (6.6). Moreover,*

- $\delta \Downarrow_* \delta[x \mapsto x^*, x' \mapsto \delta]$, *and*
- $x' \cdot (x^*) \Downarrow_* (x' \cdot (x^*)) \cdot (x^*)^* = (x' \cdot (x^*))[x \mapsto x^*, x' \mapsto x' \cdot (x^*)]$.

The proof of the latter claim uses that the rules for '$\_^*$' and '$\cdot$' satisfy the requirements of the rule format for idempotent terms. Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$x^* = (x^*)^*.$$

The perpetual loop operator discussed in Example 6.3.10 is handled in similar fashion.

Example 6.4.17 (Hiding). *Consider the hiding operator τ_I, whose rules we presented in Example 6.4.4. It is not hard to see that the rules for this operator satisfy the requirements*

in Definition 6.4.8. Indeed, since $\tau \notin I$, we have the rule

$$\frac{x \xrightarrow{\tau} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')},$$

meeting condition 3. Condition 4 is vacuously true and the others can be checked along the lines we followed in the previous examples.

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the well-known equation

$$\tau_I(x) = \tau_I(\tau_I(x)).$$

Example 6.4.18 (Idempotent relabelling)**.** *Let $h : L \to L$ be an idempotent renaming—that is, a function satisfying $h^2 = h$. (For ease of reference, we recall that, as mentioned earlier, a renaming $h : L \to L$ is idempotent if, and only if, $h(a) = a$ for each a contained in the range of h.)*

Consider the unary operation ρ_h familiar from ACP [26] and CCS [106] with rules

$$\frac{x \xrightarrow{a} x'}{\rho_h(x) \xrightarrow{h(a)} \rho_h(x')} \ (a \in L).$$

It is not hard to see that the rules for this operator satisfy the requirements in Definition 6.4.8. For example, condition 3 is met because if a is the label of the conclusion of some renaming rule, then $h(a) = a$ because h is an idempotent renaming. Condition 4 is vacuously true. Indeed, if $a \neq h(a)$, then a is not contained in the range of h, because h is an idempotent renaming, and therefore there are no a-emitting rules. The other conditions in Definition 6.4.8 can be checked along the lines we followed in the previous examples.

Therefore, Theorem 6.4.9 yields the soundness, modulo bisimilarity, of the equation

$$\rho_h(x) = \rho_h(\rho_h(x)).$$

The proviso that h be an idempotent renaming is necessary for the validity of the above equation. For instance, assume that a, b, c are pairwise different, $h(a) = b$ and $h(b) = c$. Then $\rho_h(a.\mathbf{0})$ and $\rho_h(\rho_h(a.\mathbf{0}))$ are not *bisimilar.*

6.5 Conclusions

In this study, we have presented an SOS rule format that guarantees that a unary operator is idempotent modulo bisimilarity. In order to achieve a sufficient degree of generality, that rule format relies on a companion one ensuring that certain terms are idempotent with respect to some binary operator. In addition, both rule formats make use of existing formats for other algebraic properties such as associativity [61], commutativity [117] and idempotence for binary operators [5]. In this paper, we have restricted ourselves to TSSs in GSOS format [46, 49] for the sake of simplicity. The rule formats we offered in this study can be extended to arbitrary TSSs in standard fashion, provided one gives the semantics of such TSSs in terms of three-valued stable models.

The auxiliary rule format ensuring that certain terms are idempotent with respect to some binary operator may be seen as a refinement of the one from [5]. That paper offered a rule format guaranteeing that certain binary operators are idempotent. We recall that a binary operator $\odot$ is idempotent if the equation $x \odot x = x$ holds. Of course, if a binary operation is idempotent, then any term is an idempotent for it. However, the sequential composition operator '$\cdot$' is *not* idempotent, but the term x^* is an idempotent for it. Similarly, the parallel composition operator '$\|$' is *not* idempotent, but the term $!x$ is an idempotent for $\|$. Since the laws $x^* \cdot x^* = x^*$ and $!x \,\|\, !x = !x$ play an important role in establishing, via syntactic means, that the unary Kleene star and replication operators are idempotent, we needed to develop a novel rule format for idempotent terms in order to obtain a powerful rule format for idempotent unary operations.

To our mind, idempotence of unary operators is the last 'typical' algebraic law for which it is worth developing a specialized rule format. An interesting, long-term research goal is to develop a general approach for synthesizing rule formats for algebraic properties from the algebraic law itself and some assumption on the format of the rules used to give the semantics for the language constructs in the style of SOS. We believe that this is a hard research problem. Indeed, the development of the formats for algebraic properties surveyed in [23] has so far relied on ad-hoc ingenuity and it is hard to discern some common principles that could guide the algorithmic synthesis of such formats.

6.A Proof of Theorem 6.3.7

Let R be the least reflexive relation over $\mathbb{C}(\Sigma)$ such that

- $f(p)\ R\ f(p) \odot f(p)$ and $f(p) \odot f(p)\ R\ f(p)$, for each $p \in \mathbb{C}(\Sigma)$,
- if $p \leftrightarrows p'$, $p'\ R\ q'$ and $q' \leftrightarrows q$, then $p\ R\ q$, and
- if $g \in \Sigma$ is an n-ary operator and $p_i R q_i$ for each $i \in \{1, \ldots, n\}$, then $g(p_1, \ldots, p_n) R\ g(q_1, \ldots, q_n)$.

In order to prove the theorem, it suffices to show that R is a bisimulation. To this end, note, first of all, that the relation R defined above is symmetric. Assume now that $p\,R\,q$ and $p \xrightarrow{a} p'$ for some p'. Our aim is to prove that there is some q' such that $q \xrightarrow{a} q'$ and $p'\ R\ q'$. This we show by an induction on the definition of R. Below we limit ourselves to detailing the proof for the cases when, for some $p_1 \in \mathbb{C}(\Sigma)$,

- $p = f(p_1)$ and $q = f(p_1) \odot f(p_1)$, and
- $p = f(p_1) \odot f(p_1)$ and $q = f(p_1)$.

Suppose that $f(p_1) \xrightarrow{a} p'$. Then there are a rule

$$\frac{H}{f(x) \xrightarrow{a} t}$$

and a closed substitution σ such that σ satisfies H, $\sigma(x) = p_1$ and $\sigma(t) = p'$. By condition 1 in Definition 6.3.6, there is some $\odot$-defining rule

$$\frac{H'}{x_1 \odot x_2 \xrightarrow{a} u},$$

such that

1. $H' \subseteq \{x_1 \xrightarrow{a} y_1,\ x_2 \xrightarrow{a} y_2\}$, with x_1, x_2, y_1, y_2 pairwise distinct, and
2. $t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]$.

Consider now the closed substitution

$$\sigma' = \sigma[x_1 \mapsto f(p_1), x_2 \mapsto f(p_1), y_1 \mapsto p', y_2 \mapsto p'].$$

Since $f(p_1) \xrightarrow{a} p'$, we have that σ' satisfies H'. Therefore, using the above-mentioned rule for $\odot$, we may conclude that

$$f(p_1) \odot f(p_1) \xrightarrow{a} \sigma'(u).$$

As $t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]$, we have that there are some t' and u' such that

- $t \leftrightsquigarrow t'$,
- $u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t] \leftrightsquigarrow u'$, and
- either $t' \equiv u'$, or without loss of generality $t' = C[f(t'')]$ and $u' = C[f(t'') \odot f(t'')]$, for some context $C[\ ]$ and term t''.

By Lemma 6.3.5 and the definition of R,

$$p' = \sigma(t) \leftrightarrows \sigma(t')\, R\, \sigma(u') \leftrightarrows \sigma(u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]).$$

Moreover, it is easy to see

$$\sigma(u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]) = \sigma'(u).$$

Therefore, by the definition of R, we conclude that

$$p' = \sigma(t)\, R\, \sigma'(u),$$

which was to be shown.

Assume now that $p = f(p_1) \odot f(p_1) \xrightarrow{a} p'$. Then, by condition 2 in Definition 6.3.6, there are a rule

$$\frac{\{x_i \xrightarrow{a} y_i\} \cup \{x_1 \xrightarrow{a_j} y_j \mid j \in J\} \cup \{x_2 \xrightarrow{b_k} z_k \mid k \in K\}}{x_1 \odot x_2 \xrightarrow{a} u},$$

and a closed substitution σ such that σ satisfies the premises of the rule, $\sigma(x_1) = \sigma(x_2) = f(p_1)$ and $\sigma(u) = p'$. By condition 2 in Definition 6.3.6, we have that

1. $i \in \{1, 2\}$, and
2. $vars(u) \subseteq \{x_1, x_2, y_i\}$.

Suppose, without loss of generality, that $i = 1$. Then

$$\sigma(x_1) = f(p_1) \xrightarrow{a} \sigma(y_1).$$

Therefore there are a rule

$$\frac{H}{f(x) \xrightarrow{a} t}$$

for f and a closed substitution σ' such that σ' satisfies H, $\sigma'(x) = p_1$ and $\sigma'(t) = \sigma(y_1)$. We claim that

$$\sigma'(t) = \sigma(y_1)\ R\ \sigma(u) = p'.$$

To see this, observe that, by condition 3 in Definition 6.3.6,

$$t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t].$$

Therefore, we have that there are some t' and u' such that

- $t \leftrightsquigarrow t'$,
- $u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t] \leftrightsquigarrow u'$, and
- either $t' \equiv u'$, or without loss of generality $t' = C[f(t'')]$ and $u' = C[f(t'') \odot f(t'')]$, for some context $C[\ ]$ and term t''.

By Lemma 6.3.5 and the definition of R,

$$\sigma'(t) \leftrightarrows \sigma'(t')\ R\ \sigma'(u') \leftrightarrows \sigma'(u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t]).$$

Moreover, it is easy to see

$$\sigma'(u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t]) = \sigma(u).$$

Therefore, by the definition of R, we conclude that

$$\sigma(y_1) = \sigma'(t)\ R\ \sigma(u) = p',$$

which was to be shown. Since R is symmetric, we are done.

6.B Proof of Theorem 6.4.9

Let R be the least reflexive relation over $\mathbb{C}(\Sigma)$ such that

- $f(p)\ R\ f(f(p))$ and $f(f(p))\ R\ f(p)$, for each $p \in \mathbb{C}(\Sigma)$,
- if $p \leftrightarrows p'$, $p'\ R\ q'$ and $q' \leftrightarrows q$, then $p\ R\ q$, and
- if $g \in \Sigma$ is an n-ary operator and $p_i R q_i$ for each $i \in \{1, \ldots, n\}$, then $g(p_1, \ldots, p_n) R\ g(q_1, \ldots, q_n)$.

In order to prove the theorem, it suffices to show that R is a bisimulation. To this end, note, first of all, that the relation R defined above is symmetric. Assume now that $p\,R\,q$ and $p \xrightarrow{a} p'$ for some p'. Our aim is to prove that there is some q' such that $q \xrightarrow{a} q'$ and $p'\ R\ q'$. This we show by an induction on the definition of R. Below we limit ourselves to detailing the proof for the cases when, for some $p_1 \in \mathbb{C}(\Sigma)$,

- $p = f(p_1)$ and $q = f(f(p_1))$, and
- $p = f(f(p_1))$ and $q = f(p_1)$.

Suppose that $f(p_1) \xrightarrow{a} p'$. Then, using condition 1 in Definition 6.4.8, there are a rule

$$r = \frac{H \cup \{x \overset{b_j}{\nrightarrow} \mid j \in J\}}{f(x) \xrightarrow{a} t}$$

and a closed substitution σ such that σ satisfies H, $\sigma(x) = p_1$ and $\sigma(t) = p'$. We proceed with the proof by distinguishing two cases, depending on whether rule r is a renaming rule.

Assume, first of all, that $H \subseteq \{x \xrightarrow{a} x'\}$. By conditions 5a and 5b in Definition 6.4.8,

$$t \Downarrow_f t[x \mapsto f(x), x' \mapsto t].$$

(Note that, if $H = \emptyset$, then x is the only variable that might possibly occur in t, and therefore $t[x \mapsto f(x), x' \mapsto t] = t[x \mapsto f(x)]$.)

Our goal is to use the rule r to infer a matching transition from the term $f(f(p_1))$. To this end, consider the substitution $\sigma[x \mapsto f(p_1), x' \mapsto p']$. Since $f(p_1) \xrightarrow{a} p'$, that substitution satisfies H. We claim that $\sigma[x \mapsto f(p_1), x' \mapsto p']$ also satisfies the negative premises of r. Indeed, let $j \in J$. Since σ satisfies $x \overset{b_j}{\nrightarrow}$, we have that $\sigma(x) = p_1 \overset{b_j}{\nrightarrow}$. By condition 2 in Definition 6.4.8, each b_j-emitting and f-defining rule has a positive premise of the form $x \xrightarrow{b_j} x'$. As $p_1 \overset{b_j}{\nrightarrow}$, it follows that $f(p_1) \overset{b_j}{\nrightarrow}$, as claimed. Therefore, the above rule yields

$$f(f(p_1)) \xrightarrow{a} \sigma[x \mapsto f(p_1), x' \mapsto p'](t).$$

As $t \Downarrow_f t[x \mapsto f(x), x' \mapsto t]$, we have that there are some t' and u' such that

- $t \leftrightarrow t'$,
- $t[x \mapsto f(x), x' \mapsto t] \leftrightarrow u'$, and
- either $t' \equiv u'$, or $t' = C[f(t'')]$ and $u' = C[f(f(t''))]$, for some context $C[\ ]$ and term t''.

By Lemma 6.4.7 and the definition of R,

$$p' = \sigma(t) \leftrightarrows \sigma(t') \; R \; \sigma(u') \leftrightarrows \sigma(t[x \mapsto f(x), x' \mapsto t]).$$

It is easy to see that

$$\sigma[x \mapsto f(p_1), x' \mapsto p'](t) = \sigma(t[x \mapsto f(x), x' \mapsto t]).$$

Therefore, again by the definition of R, we conclude that

$$p' = \sigma(t) \; R \; \sigma[x \mapsto f(p_1), x' \mapsto p'](t),$$

which was to be shown.

Assume now that $J = \emptyset$ and $H = \{x \xrightarrow{b} x'\}$ for some $b \neq a$. Recall that the closed substitution σ satisfies $\sigma(x) = p_1$ and $\sigma(t) = p'$. By condition 3 in Definition 6.4.8, there is an f-defining rule of the form

$$\frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} f(x')}.$$

Since, by our assumption, $f(p_1) \xrightarrow{a} p'$, instantiating the above rule with the closed substitution $\sigma' = \sigma[x \mapsto f(p_1), x' \mapsto p']$ yields that

$$f(f(p_1)) \xrightarrow{a} f(p') = f(\sigma(t)).$$

We are left to argue that $\sigma(t) = p' \; R \; f(p') = f(\sigma(t))$. To this end, observe that, by condition 5b in Definition 6.4.8, we have that

$$t \Downarrow_f f(t),$$

from which $\sigma(t) = p' \; R \; f(p') = f(\sigma(t))$ follows as above.

Assume now that $p = f(f(p_1)) \xrightarrow{a} p'$. Then, by condition 1 in Definition 6.4.8, there are a rule

$$r' = \frac{H \cup \{x \overset{b_j}{\nrightarrow} \mid j \in J\}}{f(x) \xrightarrow{a} t}$$

and a closed substitution σ such that σ satisfies H, $\sigma(x) = f(p_1)$ and $\sigma(t) = p'$.

If H is empty, then condition 1b in Definition 6.4.8 ensures that J is also empty. Thus the above rule yields the transition $f(p_1) \xrightarrow{a} \sigma[x \mapsto p_1](t)$. Moreover, by condition 5a in Definition 6.4.8,

$$t \Downarrow_f t[x \mapsto f(x)].$$

Therefore, reasoning as above,

$$\sigma[x \mapsto p_1](t)\ R\ \sigma[x \mapsto p_1](t[x \mapsto f(x)]) = \sigma(t) = p',$$

and we are done.

If $H = \{x \xrightarrow{a} x'\}$ then $f(p_1) \xrightarrow{a} \sigma(x')$. Therefore, there are some a-emitting rule for f

$$\frac{H'}{f(x) \xrightarrow{a} t'},$$

and some closed substitution σ' such that σ' satisfies H', $\sigma'(x) = p_1$ and $\sigma'(t') = \sigma(x')$. Moreover, by condition 5b in Definition 6.4.8, we have that

$$t' \Downarrow_f t[x \mapsto f(x), x' \mapsto t'].$$

Now, reasoning as above,

$$\sigma'(t')\ R\ \sigma'(t[x \mapsto f(x), x' \mapsto t']) = \sigma(t) = p'.$$

Since R is symmetric, we are done.

If $H = \{x \xrightarrow{b} x'\}$, for some $b \neq a$, then $J = \emptyset$ by condition 1 in Definition 6.4.8. Moreover $f(p_1) = \sigma(x) \xrightarrow{b} \sigma(x')$. Therefore, by condition 4 in Definition 6.4.8, there are some b-emitting rule for f

$$\frac{x \xrightarrow{b} x'}{f(x) \xrightarrow{b} t'},$$

and some closed substitution σ' such that

- $\sigma'(x) = p_1$,
- $\sigma'(t') = \sigma(x')$ and
- $\sigma'(x) = p_1 \xrightarrow{b} \sigma'(x')$.

By instantiating rule r' with the closed substitution σ', we infer that

$$f(p_1) \xrightarrow{a} \sigma'(t).$$

Now, by condition 5c in Definition 6.4.8, we have that

$$t \Downarrow_f t[x \mapsto f(x), x' \mapsto t'].$$

Therefore, reasoning as above,

$$\sigma'(t)\ R\ \sigma'(t[x \mapsto f(x), x' \mapsto t']) = \sigma(t) = p'.$$

Since R is symmetric, we are done.

Chapter 7

PREG Axiomatizer – A Ground Bisimilarity Checker for GSOS with Predicates

7.1 Introduction

Proving that two process terms are related by some notion of behavioural equivalence is at the heart of the equivalence-checking approach to verification. In this chapter we introduce a tool named PREG Axiomatizer[1] that tackles this problem focusing on ground (*i.e.*, fully specified) terms built using operations defined using the *preg* format, a *pr*edicates *e*xtension of the *G*SOS format presented in Chapter 3. GSOS [49] is a restricted, yet powerful, way of defining Structural Operational Semantics (SOS) for programming and specification languages in the style introduced by Plotkin in [124]. We refer the reader to Chapter 3 for the detailed description and intuition behind the *preg* rule format and the considered notion of behavioural equivalence, which is a natural extension to predicates of the classic strong bisimulation equivalence.

Building on the techniques in [6, 35], we proposed in Chapter 3 a procedure to construct a finite collection of sound equations that can be used to bring any ground term to a normal form. We showed that the normal forms of two terms are equal if and only if the terms are bisimilar. Given a set of actions $\mathcal{A}$ and a set of predicates $\mathcal{P}$, the normal forms we refer to are terms built according to the

[1] The tool is downloadable from `http://goriac.info/tools/preg-axiomatizer/`.

grammar for finite trees with predicates, namely

$$s ::= \delta \mid \kappa_P \ (\forall P \in \mathcal{P}) \mid a.s \ (\forall a \in \mathcal{A}) \mid s + s,$$

that are of the shape $t = \sum_{i \in I} a_i.t_i + \sum_{j \in J} \kappa_{P_j}$. Here the P_j's are all the predicates satisfied by t, and the t_i's are terms in normal form. The empty sum ($I = \emptyset, J = \emptyset$) is denoted by the constant δ.

Intuitively, δ represents the process exhibiting no behaviour, $s + t$ is the nondeterministic choice between the behaviours of s and t, while $a.t$ is a process that first performs action a and behaves like t afterwards. For each predicate P we consider a constant κ_P, which denotes a process with no transitions. This process only satisfies P. A finite tree satisfies predicate P if and only if it has κ_P as a top summand of its associated normal form. We refer to predicates in $\mathcal{P}$ as *existential predicates*. The operational semantics that captures this intuition is given by the rules of BCCSP extended with predicates. The SOS specification for this language consists of rules parameterized over all actions a and explicit predicates P:

$$\frac{}{a.x \xrightarrow{a} x}, \quad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}, \quad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}, \quad \frac{}{P\kappa_P}, \quad \frac{Px}{P(x + y)}, \quad \frac{Py}{P(x + y)}.$$

In Chapter 3 we showed that, for the above language, the following set of axioms [106] is sound and ground-complete for bisimilarity on the set of ground finite trees with predicates:

$$\begin{array}{lll} x + y = y + x & & (x + y) + z = x + (y + z) \\ x + x = x & & x + \delta = x \end{array}$$

Recall that our purpose is to find ground-complete axiomatizations like the one above for all the languages given in the *preg* format. In order to achieve this goal for operators whose rules involve negative premises, we use the *restriction operator* $\partial_{\mathcal{B},\mathcal{Q}}$ (where $\mathcal{B} \subseteq \mathcal{A}$ and $\mathcal{Q} \subseteq \mathcal{P}$ are the sets of initially forbidden actions and predicates, respectively). The semantics of $\partial_{\mathcal{B},\mathcal{Q}}$ is given by the following two types of transition rules:

$$\frac{x \xrightarrow{a} x'}{\partial_{\mathcal{B},\mathcal{Q}}(x) \xrightarrow{a} \partial_{\emptyset,\mathcal{Q}}(x')} \text{ if } a \notin \mathcal{B}, \quad \frac{Px}{P(\partial_{\mathcal{B},\mathcal{Q}}(x))} \text{ if } P \notin \mathcal{Q}.$$

The axiomatization of the operators $\partial_{\mathcal{B},\mathcal{Q}}$ is provided in Chapter 3.

Internally, PREG Axiomatizer brings the provided rule system to a "manageable" format, introducing auxiliary operators as described in Chapter 3, and afterwards performs the axiomatization itself. The tool is implemented in the Maude lan-

guage [59], which has been already proven to be very useful for analyzing SOS rule formats in [114, 57]. Not only did we use Maude as a programming language, but also as an equational reduction system for the generated sets of axioms.

Contribution PREG Axiomatizer is, to our knowledge, the first public tool that automatically derives sound and ground-complete axiomatizations modulo bisimilarity for GSOS-like languages. Prior to using the techniques presented in [6, 35] and Chapter 3, one had to use ingenuity and dedicate a considerable amount of time in order to obtain axiomatizations for a language with even a limited number of operators.

The tool is generic, in the sense that the SOS specification defining the labelled transition system semantics of the process calculus is provided by the user. One does this in terms of *well-founded* GSOS systems, which only allow for the derivation of finite labelled transition systems for the given terms (see [6] for more details). As presented in [53], the generated axiomatizations are guaranteed to be confluent, but, as a downside of our approach, only weakly normalizing. This downside is diminished by the fact that there exists a substantial decidable subclass of systems, namely the linear and syntactically well-founded ones [53], for which the generated axiomatizations are strongly normalizing. This subclass includes important languages such as CCS, CSP, and ACP.

7.2 Case Studies

In this section we present two scenarios involving several classic operations with their semantics extended with certain explicit predicates. Conventionally, the tool language accepts process term variables such as `X`, `X1`, `Y'`, actions like `a`, `b`, `c`, `a[0]`, `b[2]`, `c["name"]`, and predicates like `P`, `Q`, `P[1]`, `Q["prop"]`.

Example 7.2.1. *Let us describe how PREG Axiomatizer is used in order to prove that "$a.(a.\kappa_{\downarrow}; b.(a.\kappa_{\downarrow}))$" and "*`while` $a.b.\kappa_{\downarrow}$ `do` $a.\kappa_{\downarrow}$*" are bisimilar. Here* `_;_` *and* `while_do_` *are, respectively, the sequential composition and the process loop operators (presented in [49]) extended to the* preg *format with the immediate successful termination predicate* $\downarrow$ *(which we choose to denote by* `P` *in the specification for tool consumption). In Figure 7.1 we present the operational semantics for these operations with the rules given both in standard notation, as well as using the syntax supported by the tool.*

$$\frac{x \xrightarrow{a} x'}{x;y \xrightarrow{a} x';y} \quad : \quad \texttt{X -(a)-> X'} \;\texttt{===}\; \texttt{X ; Y -(a)-> (X' ; Y)}$$

$$\frac{x\downarrow \;\; y \xrightarrow{a} y'}{x;y \xrightarrow{a} y'} \quad : \quad \texttt{P(X) , Y -(a)-> Y'} \;\texttt{===}\; \texttt{X ; Y -(a)-> Y'}$$

$$\frac{x\downarrow \;\; y\downarrow}{(x;y)\downarrow} \quad : \quad \texttt{P(X) , P(Y)} \;\texttt{===}\; \texttt{P(X ; Y)}$$

$$\frac{x\downarrow}{(\texttt{while}\ x\ \texttt{do}\ y)\downarrow} \quad : \quad \texttt{P(X)} \;\texttt{===}\; \texttt{P(while X do Y)}$$

$$\frac{x \xrightarrow{a} x'}{\texttt{while}\ x\ \texttt{do}\ y \xrightarrow{a} y;\texttt{while}\ x'\ \texttt{do}\ y} \quad : \quad \texttt{X -(a)-> X'} \;\texttt{===}\; \texttt{while X do Y -(a)-> Y ; while X' do Y}$$

Figure 7.1: *preg* rule system for _;_ and `while_do_`

The rules involving action a also have to be instantiated for b. After providing this specification, the user can press the button labelled "Axiomatize" and the tool generates a Maude specification including the axioms obtained by following the procedure described in Chapter 3. We exemplify a small part of the output which consists of the axiomatization for the `while_do_` *operator:*

```
  eq while X0 + X1 do X2 = while X0 do X2 + while X1 do X2 .
 ceq while X do Y = a . (Y ; while X' do Y) if a . X' := X .
 ceq while X do Y = b . (Y ; while X' do Y) if b . X' := X .
 ceq while X do Y = k[P] if X := k[P] .
  eq while X1 do X2 = delta [owise] .
```

In order to check for the bisimilarity of the two process terms introduced at the beginning of the current example, one loads the generated specification and uses the Maude command `reduce`*:*

```
> reduce a . (a . k[P] ; b . (a . k[P])) ==
         while a . b . k[P] do a . k[P] .
result Bool: true

> reduce while a . b . k[P] do a . k[P] .
result PTerm: a . a . a . b . a . a . k[P]
```

We successfully used PREG Axiomatizer to further extend the operational semantics of _;_ with the predictable non-failure predicate $\neq \delta$ (which plays the role of the predicate "$\neq 0$" presented in [10]) with the rules: $\frac{x \xrightarrow{a} x' \;\; y \neq \delta}{x;y \xrightarrow{a} x';y}$ $\frac{x \neq \delta \;\; y \neq \delta}{(x;y) \neq \delta}$. We managed to test the property that $x;\delta$ and δ are bisimilar on various closed instantiations.

It is worth noting that this property does not always hold for the initial version of _;_.

Example 7.2.2. *In this example we show how we use our tool to obtain the execution tree of a network of communicating processes. This procedure is useful, for instance, when one needs to use an external model checker to verify if the communication protocol satisfies certain logical properties. Our example is based on a case study from [25].*

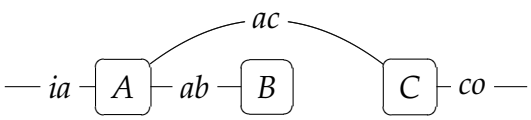


Figure 7.2: Communication protocol

Consider the process network given in Figure 7.2 where A, B, C are the communicating processes and ia, ab, ac, co are the ports. The actions of sending, receiving, and synchronizing on the datum d over the port p are denoted by, respectively, $p!d$, $p?d$, and $p\#d$. By using these actions, the parallel composition operator _||_, and the immediate successful termination predicate $\downarrow$, we specify the whole protocol as the term:

$$T = ia?d \,.\, (ab!d \,.\, \kappa_{\downarrow} \parallel ac!d \,.\, \kappa_{\downarrow}) \parallel ab?d \,.\, \kappa_{\downarrow} \parallel ac?d \,.\, co!d \,.\, \kappa_{\downarrow} \,.$$

We present preg *rules for _||_, in which $act \in \{p!d, p?d, p\#d\}$:*

$$\frac{x \xrightarrow{act} x'}{x \parallel y \xrightarrow{act} x' \parallel y} \qquad \frac{y \xrightarrow{act} y'}{x \parallel y \xrightarrow{act} x \parallel y'} \qquad \frac{x \downarrow \;\; y \downarrow}{(x \parallel y) \downarrow}$$

$$\frac{x \xrightarrow{p!d} x' \;\; y \xrightarrow{p?d} y'}{x \parallel y \xrightarrow{p\#d} x' \parallel y'} \qquad \frac{x \xrightarrow{p?d} x' \;\; y \xrightarrow{p!d} y'}{x \parallel y \xrightarrow{p\#d} x' \parallel y'}$$

Figure 7.3: *preg* rule system for _||_

The input for PREG Axiomatizer *consists of:*

- *the predicate rule* $\dfrac{\texttt{P(X)} \;\; , \;\; \texttt{P(Y)}}{\texttt{P(X || Y)}}$,
- *all the instantiations of the first two transition rules in Figure 7.1 for which act is an action from the set $\mathcal{A} = \{ia?d, ab!d, ac!d, ab?d, ac?d, ab\#d, ac\#d, co!d\}$* $\left(\text{e.g.,} \quad \dfrac{\texttt{X -(a["ia?d"])-> X'}}{\texttt{X || Y -(a["ia?d"])-> X' || Y}} \right)$, *and*

- *all the instantiations of the last two transition rules in Figure 7.1 in which p is a port from* $\{ab, ac\}$ $\left(\text{e.g.,} \begin{array}{c} \texttt{X -(a["ab?d"])-> X' , Y -(a["ab!d"])-> Y'} \\ \texttt{===} \\ \texttt{X || Y -(a["ab\#d"])-> X' || Y'} \end{array} \right)$.

We generate the process network execution tree (consisting of 582 states) by calling the command `reduce` *on the specification term T:*

```
> reduce ((a["ia?d"] . (a["ab!d"] . k[P] || a["ac!d"] . k[P])) ||
          a["ab?d"] . k[P]) || a["ac?d"] . a["co!d"] . k[P] .
result PTerm: a["ab?d"] . (...) + a["ac?d"] . (...) + a["ia?d"] . (...)
```

The parallel composition allows for arbitrary interleavings of the actions in $\mathcal{A}$*, but it does not enforce the communication over the ports ab and ac. Hiding these ports so that other processes cannot interfere with the internal communications is desirable. This can be done with the help of a generalization of the* restriction operator $\partial_{\mathcal{B},\mathcal{Q}}$ *presented in Section 7.1, denoted by* $\overline{\partial}_{\mathcal{B},\mathcal{Q}}$*, that preserves the imposed restrictions on actions throughout the whole computation, not only for the first step. Forbidding independent send and receive actions over the ports ab and ac is denoted by the term* $\overline{\partial}_{\{p!d,p?d \mid p\in\{ab,ac\}\},\emptyset}(T)$*. In PREG Axiomatizer we use* `%%[`$\mathcal{B}$`;`$\mathcal{Q}$`]` *as a syntactic notation for* $\overline{\partial}_{\mathcal{B},\mathcal{Q}}$*:*

```
> reduce %%[ a["ab?d"] a["ab!d"] a["ac?d"] a["ac!d"] ; empty ](
          (( a["ia?d"] . (a["ab!d"] . k[P] || a["ac!d"] . k[P])) ||
             a["ab?d"] . k[P]) || a["ac?d"] . a["co!d"] . k[P]) .
result PTerm: a["ia?d"] . (a["ab#d"] .  a["ac#d"] . a["co!d"] . k[P] +
                           a["ac#d"] . (a["ab#d"] . a["co!d"] . k[P] +
                                        a["co!d"] . a["ab#d"] . k[P]))
```

We also tested our tool by generating the normal form of $a^3.\kappa_{\downarrow} \parallel b^3.\kappa_{\downarrow} \parallel c^3.\kappa_{\downarrow}$ *and obtained the same "2 page long" execution tree showed in [28], consisting of 6927 states. Maude derives this execution tree in less than 500 milliseconds on a machine with a 2.53GHz processor and 4GB of RAM.*

Example 7.2.3. *We now show how PREG Axiomatizer is used in order to perform equational proofs when working with predicates that have implicit behaviour. Consider, for instance, the case of the* eventual successful termination predicate $\lightning$*. It represents the extension of* $\downarrow$*, introduced in the previous examples, with the requirement that if* $t \lightning$ *holds for a term t, then* $a.t \lightning$ *holds for any action a.*

Recall from Section 7.1 that our approach is based on denoting the property $\lightning$ *by using the explicit process constant* $\kappa_{\lightning}$ *as a summand of the analyzed term. The above characterization of* $\lightning$ *is given by the axiom* $a.(t + \kappa_{\lightning}) = a.(t + \kappa_{\lightning}) + \kappa_{\lightning}$*. With this in mind, one could check, for instance, if a process t "eventually terminates" by checking if it is bisimilar to* $t + \kappa_{\lightning}$*.*

In order to prove that $a.\kappa_{\lightning} \parallel b.\kappa_{\lightning}$ *is bisimilar to* $(a.\kappa_{\lightning} \parallel b.\kappa_{\lightning}) + \kappa_{\lightning}$ *we need to let the tool "know" that it should treat* $\lightning$ *(denoted by* `Q` *in the specification) as an implicit predicate*

by using the operation `expandImplicit`*. This operation receives a term and the set of implicit predicate names:*

```
> reduce expandImplicit(a . k[Q] || b . k[Q], Q) ==
         expandImplicit(a . k[Q] || b . k[Q] + k[Q], Q) .
result Bool: True

> reduce expandImplicit(a . k[Q] || b . k[Q], Q) .
result PTerm: k[Q] + a . (k[Q] + b . k[Q]) + b . (k[Q] + a . k[Q])
```

Predicates with implicit behaviour, like $\lightning$*, can only be used during the normalization process if the operators whose definition involves these predicates are given by rules that satisfy certain sanity constraints mentioned in Chapter 3. The tool does not currently support the automated checking for those constraints, so the user needs to do it manually before using the feature presented above. The parallel composition operator does meet those constraints.*

7.3 Discussion and Future Work

Aside from the features mentioned in Section 7.2, an important part of the PREG Axiomatizer engine is dedicated to checking for the conformance of specified operations and rules to the various formats presented in Chapter 3.

There are many areas in which the tool and the theory behind it can be improved. First and foremost, an important feature would be to allow the user to specify guarded recursively defined terms therefore greatly increasing the complexity of the case studies our tool can handle. The most natural way to extend our approach in order to reason about the bisimilarity of such terms is to integrate the technique presented in [6], which is also based on generating complete axiomatizations for a class of GSOS languages generating regular behaviours. The main difficulty of this task will be the search for good strategies for applying the axioms and the unique fixed-point induction rule.

Another tool development direction is concerned with the ability to automatically check if the specification meets certain complex requirements. One of these requirements is, as presented in Section 7.1, the syntactic well-foundedness of the given system. Without this feature the user needs to be careful not to specify operators such as the *reentrant server* $!\_$, defined by the rule $\dfrac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \parallel !x}$, for which non-normalizing axioms are derived: $!x = !'(x, x)$, $!'(a.x', x) = a.(x' \parallel !x)$.

Another requirement the tool could check for consists of the sanity constraints we mentioned in Example 7.2.3.

Chapter 8

Meta SOS – A Maude Based SOS Meta-Theory Framework

8.1 Introduction

Structural Operational Semantics [122] is a well known approach for intuitively specifying the semantics of programming and specification languages by means of rules. These rules can be analyzed using meta-theoretic results in order to infer certain properties about language constructs by purely syntactic means. Research on SOS meta-theory has at its core the development of rule formats that, if respected, will guarantee that some language constructs have certain properties, such as commutativity, associativity, and idempotence. We refer the reader to [23] for an overview on how to derive these properties as well as axiomatizations. Rule formats can also be used to obtain congruence properties for behavioural equivalences (see, e.g., [14]) and semantic properties such as determinism of transition relations [5].

Despite the large body of research on the meta-theory of SOS, to the best of our knowledge, there currently does not exist an extensible software tool integrating the results obtained so far in that research area. (We briefly review some of the existing software tools below.) This is an unsatisfactory state of affairs since such a software framework would allow language designers to benefit from the results in the meta-theory of SOS while experimenting with their language designs. The design of programming and specification languages is a highly non-trivial endeavour and tool support is needed in order to support prototyping of language designs, their algorithmic analysis and early checking of desired semantic

properties. The meta-theory of SOS provides, for example, syntactic criteria guaranteeing the validity of semantic properties, but checking such criteria by hand is error prone and quickly becomes infeasible.

Contribution In this paper we introduce Meta SOS[1], a framework for handling SOS specifications, with the purpose of performing simulations, deriving axiomatizations, and checking for rule formats. Though it has a different line of implementation, Meta SOS continues the work we started with a prototype named PREG Axiomatizer (see Chapter 7), dedicated to deriving axiom systems from SOS specifications.

We are aware of other software tools that are somewhat related to Meta SOS. In [114] the authors show how to prototype SOS meta-theory in Maude [59]. That paper was a good point of reference for us both for implementation details and future work ideas. The Process Algebra Manipulator (PAM) [99] is designed to perform algebraic reasonings on programs written only in CCS [106], CSP [95] and LOTOS [54]. PAM does not allow the user to define their own language. The Maude MSOS Tool (MMT) [57] does provide this facility, however it does not focus on axiomatizations or rule formats, and, unfortunately, neither does it facilitate a natural extension with new features. LETOS [87] is a lightweight tool to aid the development of operational semantics, which supports execution and tracing, as well as quality rendering using LaTeX. LETOS makes some first steps towards checking operational conservativity along the lines proposed in the paper [86].

Structure The rest of the paper is organized as follows. In Section 8.2 we present some preliminaries on SOS, Maude and Meta SOS. Section 8.3 describes the three components the framework currently provides: a simulator and bisimilarity checker (Section 8.3.1), a sound and ground-complete axiom schema deriver (Section 8.3.2), and a commutativity format checker (Section 8.3.3). It also includes Section 8.3.4, where we present a case study that integrates all the previously mentioned components, and Section 8.3.5, where we briefly show how to extend the framework with more functionalities. Finally, Section 8.4 concludes the paper and points out possible directions for future research.

[1] The framework is downloadable from `http://goriac.info/tools/meta-sos/`.

8.2 Preliminaries

Maude [59] is a high-level language providing support for specifying multi-sorted signatures, equational and rewrite theories. Not only is it an excellent environment to perform reasonings with these theories at object level, but also, due to its reflective capabilities, to analyze and operate with them at meta-level. Previous efforts from [57, 114, 130, 137] and Chapter 7 have shown its suitability in facilitating SOS specifications.

Meta SOS is implemented in Maude as a metalanguage application [83]. This means that the framework extends Maude with capabilities such as providing SOS specifications and operating with them. After opening the Maude environment and loading the framework by using the command `load metasos.maude`, a specification is given using the standard syntax for inputting functional modules: `(fmod SPECIFICATION is ... endfm)`, where "`...`" consists of constructs that are discussed in the remainder of the paper.

We assume a *signature* Σ, which is a set of function symbols with fixed arities (typical members: f, g). Function symbols with arity 0 are referred to as *constants*. Moreover, we assume an infinite set of *variables* V (typical members: x, y).

Open terms are inductively built using variables and function symbols by respecting their arities. The set of open terms is denoted by $\mathbb{T}(\Sigma)$ (typical members: s, t). By $T(\Sigma)$ we denote the set of terms formed without variables, referred to as *closed terms* (typical members: p, q). *Substitutions*, which are functions of the type $\sigma : V \to \mathbb{T}(\Sigma)$, have the role of replacing variables in an open term with other (possibly open) terms.

Meta SOS has a basic set of sorts. One of them represents the domain of process terms $\mathbb{T}(\Sigma)$ and has the name `PTerm`. It is important to note that we did not use the name `Term` due to it being reserved for operating at meta-level with general terms formed using Maude multi-sorted signatures. In order to have access to the sort `PTerm` one needs to include a core Meta SOS module named `RULES` in the specification: `including RULES .` Operations are given using a standard syntax. For instance, the following construct declares a binary operation f over process terms: `op f : PTerm PTerm -> PTerm [metadata "sos"] .` Notice the use of the attribute in square brackets, which makes it possible for f to be used in SOS specifications. Variables are also given using a standard syntax: `var x y : PTerm .`

8.2.1 Transition System Specifications in Meta SOS

We will now describe how transition system specifications are expressed in Meta SOS.

Definition 8.2.1 (Transition System Specification)**.** *Consider a signature Σ and a set of labels L (with typical members l, l'), $t, t' \in \mathbb{T}(\Sigma)$ and $l \in L$. A* positive transition formula *is a triple (t, l, t'), written $t \xrightarrow{l} t'$, with the intended meaning: process t performs the action labelled as l and becomes process t'. A* negative transition formula *is a tuple (t, l), written $t \xrightarrow{l} \!\!\!\!\!\rightarrow$, with the meaning that process t cannot perform the action labelled as l.*

A transition rule *is a pair (H, α), where H is a set of formulae and α is a formula. The formulae from H are called* premises *and the formula α is called the* conclusion*. A transition rule is often denoted by $\dfrac{H}{\alpha}$ and has the following generic shape:*

$$\frac{\{t_i \xrightarrow{l_i} t_i' \mid i \in I\} \cup \{t_j \xrightarrow{l_j} \!\!\!\!\!\rightarrow \mid j \in J\}}{t \xrightarrow{l} t'},$$

where I, J are index sets, $t, t', t_i, t_i', t_j \in \mathbb{T}(\Sigma)$, and $l_i, l_j \in L$. A transition system specification *(abbreviated TSS) is a triple $(\Sigma, L, \mathcal{R})$ where Σ is a signature, L is a set of labels, and $\mathcal{R}$ is a set of transition rules of the provided shape.*

In Meta SOS, positive and negative formulae are denoted by expressions such as `t -(l)-> t'` and `t -(l)/>`, respectively. Here `t`, `t'` are variables of sort `PTerm` and `l` is a variable of another provided sort, `PLabel`. A transition rule $\dfrac{H}{c}$ is declared as `H === c`, where `H` consists of a (possibly empty) list of comma-separated formulae. The entire set of transition rules is given as a list of rules wrapped in a Maude membership axiom declaration: `mb ... : PAllRules .`

To exemplify a full Meta SOS specification, consider the BCCSP system from [79]. Its signature Σ_{BCCSP} includes the *deadlock* process $\mathbf{0}$, a collection of *prefix* operators $l.\_$ $(l \in L)$ and the binary *choice* operator $\_ + \_$. For a fixed $L = \{a, b, c\}$, the deduction rules for these operators are:

$$\frac{}{l.x \xrightarrow{l} x} \qquad \frac{x \xrightarrow{l} x'}{x + y \xrightarrow{l} x'} \qquad \frac{y \xrightarrow{l} y'}{x + y \xrightarrow{l} y'}, \text{ where } l \in L.$$

```
(fmod SPECIFICATION is including RULES .
  op 0   : -> PClosedTerm .
  op _._ : PLabel PTerm -> PTerm
           [metadata "sos"] .
  op _+_ : PTerm PTerm -> PTerm
           [metadata "sos"] .

  ops a b c : -> PAction .

  var x y x' y' : PTerm .
  var l : PLabel .
```

```
  mb
  ===
  l . x -(l)-> x

  x -(l)-> x'
  ===
  x + y -(l)-> x'

  y -(l)-> y'
  ===
  x + y -(l)-> y' : PAllRules .
endfm)
```

As illustrated, Maude provides good support for working with operators in infix notation. As an improvement over PREG Axiomatizer (see Chapter 7), we use the generic label `l` of sort `PLabel` as syntactic sugar, instead of writing rules for each of the three concrete actions. These actions are declared as constants – operations without a domain, of sort `PAction`, which is a subsort of `PLabel` described later starting with Section 8.3.1. (We can have other types of labels, not just actions.) The deadlock process is also declared as a constant of sort `PClosedTerm`, which stands for $T(\Sigma)$ and is declared internally as a subsort of `PTerm`.

8.3 Meta SOS Components

Though Meta SOS is conceived as a general SOS framework, we have so far limited our development to case studies involving only GSOS systems [49]. These systems have certain desirable properties and, in spite of their restricted format, they cover most of the operations in the literature [14].

Definition 8.3.1 (GSOS rule format)**.** *Consider a process signature Σ. A* GSOS rule *ρ over Σ has the shape:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, j \in I_i\} \cup \{x_i \xrightarrow{l_{ij}} \mid i \in J, j \in J_i\}}{f(\vec{x}) \xrightarrow{l} C[\vec{x}, \vec{y}]},$$

where all variables are distinct, f is an operation symbol from Σ with arity n, $I, J \subseteq \{1, \ldots, n\}$, I_i, J_i are finite index sets, the l_{ij}'s and l are labels standing for actions ranging over the set L, and $C[\vec{x}, \vec{y}]$ is a Σ-context with variables including at most the x_i's and y_{ij}'s.

A GSOS system *is a TSS $(\Sigma, L, \mathcal{R})$ such that Σ and L are finite, and $\mathcal{R}$ is a finite set of rules in the GSOS format.*

The operational semantics of a TSS in the GSOS format is given in terms of a labelled transition system (LTS), whose transition relations are defined by structural induction over closed terms using the rules. An essential property we make use of is that LTS's induced by GSOS systems are finitely branching [49]. It is easy to see that BCCSP respects the GSOS format.

In what follows we present three main features provided by the Meta SOS framework.

8.3.1 Simulator and Bisimilarity Checker

The purpose of the simulator associated to a TSS is to find all transitions for a given closed term. Formally, the simulator finds, for a given closed term p, all the labels l and closed terms p' such that $p \xrightarrow{l} p'$. This is a slightly more general approach than the one in [114], where the user needs to give not only the initial term, but also the label as input.

To illustrate how to use the simulator, consider $\_ \parallel \_$, the interleaving parallel composition without communication. We want to also define its behaviour in the context of the termination predicate $\downarrow$. As Meta SOS does not currently provide direct support for working with predicates, unlike PREG Axiomatizer, we model predicate satisfiability by means of transitions. By adding the termination predicate trigger as a label, the rules for the prefix and choice operators remain the same. The rules for interleaving parallel composition are:

$$\frac{x \xrightarrow{\alpha} x'}{x \parallel y \xrightarrow{\alpha} x' \parallel y} \qquad \frac{y \xrightarrow{\alpha} y'}{x \parallel y \xrightarrow{\alpha} x \parallel y'} \qquad \frac{x \xrightarrow{\downarrow} x' \;\; y \xrightarrow{\downarrow} y'}{x \parallel y \xrightarrow{\downarrow} \mathbf{0}}.$$

Here α stands for any of the considered actions from the set $\{a, b, c\}$, but not for the termination predicate trigger. Also, we want to make sure that the last rule is applied only for $\downarrow$, but for none of the actions. To specify this we enhance the previous specification with a new sort for predicates as a subset of labels, add the termination predicate, a variable ranging only over actions, and the rules:

```
sort PPredicate .
subsort PPredicate < PLabel .

x -(alpha)-> x'
===
x || y -(alpha)-> x' || y

x -(|)-> x' , y -(|)-> y'
===
x || y -(|)-> 0
```

```
op | : -> PPredicate .
var alpha : PAction .

y -(alpha)-> y'
===
x || y -(alpha)-> x || y'
```

We first need to set up a simulator `(derive simulator SPECIFICATION .)`. Not only does this prepare the metalanguage application to perform simulations, but also outputs a pure Maude specification that can be used outside the Meta SOS environment for simulations within the specified system. The advantage of using this generated simulator is a minor gain in performance due to the elimination of the overhead that comes with any metalanguage application. In addition, this allows for the use of Maude tools such as the reachability analyzer and the LTL model checker.

To perform a one step simulation for a given term we use the command `(simulate ... .)`. For instance, the concrete call to observe how $p = \downarrow .\mathbf{0} \parallel a.\mathbf{0}$ is simulated is `(simulate | . 0 || a . 0 .)`. The output is a list of pairs of the shape $< l \# p' >$, where l is a label of a provable transition and p' is the resulting term. In our case the output is: `Possible steps: < a # | . 0 || 0 >`. Note that due to our making a clear distinction between actions and predicates only one of the rules involving actions is applicable. The term $(\downarrow .\mathbf{0} + b.\mathbf{0}) \parallel (c.\mathbf{0} + \downarrow .\mathbf{0})$, on the other hand, does involve all the specified rules:

```
> (simulate | . 0 + b . 0 || c . 0 + | . 0 .)

Possible steps:
  < b # 0 || c . 0 + | . 0 >
  < c # | . 0 + b . 0 || 0 >
  < | # 0 >
```

From the implementation perspective we tackled one of the issues suggested as future work in [114]. The caveat of the tool presented in that paper is that the user needs to provide term matching and substitution definitions by hand for every operator. Our approach uses and extends Maude's meta-level functionality of working with substitutions in such a way that it becomes transparent to the user.

As the idea of rewrite-based SOS simulators has already been explored in [57, 114, 137], we focused only on performing one step simulations. Having that

functionality, it was natural to derive a strong bisimilarity checker that implements the following definition.

Definition 8.3.2 (Strong Bisimilarity [120])**.** *Consider a TSS* $\mathcal{T} = (\Sigma, L, \mathcal{R})$*. A relation* $R \subseteq T(\Sigma) \times T(\Sigma)$ *is a* strong bisimulation *if and only if it is symmetric and*

$$\forall_{p,q}(p,q) \in R \Rightarrow (\forall_{l,p'} p \xrightarrow{l} p' \Rightarrow \exists_{q'} q \xrightarrow{l} q' \wedge (q,q') \in R).$$

Two closed terms p *and* q *are* strongly bisimilar*, denoted by* $p \leftrightarrows^{\mathcal{T}} q$*, if there exists a strong bisimulation relation* R *such that* $(p,q) \in R$*. Whenever* $\mathcal{T}$ *is known from the context, we simply write* $p \leftrightarrows q$*.*

In [49] it is shown that bisimilarity is a congruence for GSOS systems and that the labelled transition systems defined using these systems are finitely branching. These properties are necessary when checking for strong bisimilarity by means of the axiom schema that we will present in Section 8.3.2. Before that, let us present how to check strong bisimilarity using Meta SOS.

In order to check if, for instance, $a.\mathbf{0} \parallel b.\mathbf{0} \leftrightarrows a.b.\mathbf{0} + b.a.\mathbf{0}$ holds, we use the command

```
> (check (a . 0 || b . 0) ~ (a . b . 0 + b . a . 0) .)

result: true.Bool
```

The algorithm is a straightforward Maude implementation of Definition 8.3.2.

The Maude specification output when setting up the simulator also includes the bisimilarity checker. Running this specification allows one to directly use the functions that implement the simulator and bisimilarity checker features, which have the same name as the user interface commands. These are called using `reduce` in the core Maude environment: `reduce simulate ... .` and `reduce check ... ~ ... .`

The bisimilarity checker does not currently handle process terms with infinite behaviour. The module presented in the next section, however, can check if two terms from Σ_{BCCSP}, defined using guarded recursion, are bisimilar.

8.3.2 Axiom Schema Deriver

As an alternative method for reasoning about strong bisimilarity, Meta SOS includes a component for generating axiom schemas that are sound and ground-

complete modulo bisimilarity. There has been a notable amount of effort put into developing algorithms for axiomatizations for GSOS-like systems [6, 35] (also see Chapter 3), yet all involve several transformations of the original system before deriving the axioms. After implementing one such algorithm in the tool PREG Axiomatizer (see Chapter 7), a simpler method was developed in Chapter 4. We slightly adapt that approach here by using an extended version of the prefix operation and by also showing how to axiomatize operations defined using rules with negative premises, not just positive ones.

When given a signature Σ that includes Σ_{BCCSP}, the purpose of an axiomatization of strong bisimilarity is to rewrite each term $t \in T(\Sigma)$, that is semantically well founded in the sense of Definition 5.1 from [6], to another term t' such that $t \leftrightarrow t'$ and $t' \in T(\Sigma_{\text{BCCSP}})$. This reduces the problem of axiomatizing bisimilarity over $T(\Sigma)$ to that of axiomatizing it over BCCSP. It is well known [92] that the following axiomatization (denoted by E_{BCCSP}) is sound and ground-complete for bisimilarity on BCCSP:

$$x + y = y + x \qquad x + x = x$$
$$(x + y) + z = x + (y + z) \qquad x + \mathbf{0} = x$$

In order to set up Maude to perform equational reasoning using E_{BCCSP}, we can declare that `_+_` is associative and commutative so that rewrites are performed modulo these two properties: `op _+_ : PTerm PTerm -> PTerm [assoc comm metadata "sos"] .` Also, even though we could specify idempotence and identity element as attributes, for performance reasons we add the last two equations explicitly to the specification: `eq x + x = x . eq x + 0 = x .` For convenience Meta SOS already includes a module with the signature and equations for BCCSP named `ET-BCCSP` that can be included in the specification. For this reason, the names `.` and `+` are reserved, which means that if the user wants to specify his/her own version of the prefix and choice operations some other names need to be used.

Definition 8.3.3 (Head Normal Form)**.** *Let Σ be a signature such that $\Sigma_{\text{BCCSP}} \subseteq \Sigma$. A term t in $\mathbb{T}(\Sigma)$ is in* head normal form *(for short, h.n.f.) if $t = \sum_{i \in I} l_i.t_i$. The empty sum ($I = \emptyset$) is denoted by the deadlock constant $\mathbf{0}$.*

Definition 8.3.4 (Disjoint extension)**.** *A GSOS system G' is a disjoint extension of a GSOS system G, written $G \sqsubseteq G'$, if the signature and the rules of G' include those of G, and G' does not introduce new rules for operations in G.*

Definition 8.3.5 (Axiomatization schema)**.** *Let $\mathcal{T} = (\Sigma, L, \mathcal{R})$ be a TSS in GSOS format such that* BCCSP $\sqsubseteq \mathcal{T}$*. By $E_{\mathcal{T}}$ we denote the axiom system that extends E_{BCCSP}*

with the following axiom schema for every operation f in $\mathcal{T}$, parameterized over the vector of closed process terms $\vec{p}$ in h.n.f.:

$$f(\vec{p}) = \sum \left\{ l.C[\vec{p},\vec{q}] \;\middle|\; \rho = \frac{H}{f(\vec{x}) \xrightarrow{l} C[\vec{x},\vec{y}]} \in \mathcal{R}, \vec{p} = \sigma(\vec{x}), \vec{q} = \sigma(\vec{y}) \text{ and } \checkmark(\vec{p},\rho) \right\},$$

where $\checkmark$ *is defined as* $\checkmark(\vec{p},\rho) = \bigwedge_{p_k \in \vec{p}} \checkmark'(p_k, k, \rho)$,

and $$\checkmark'\left(p_k, k, \frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, j \in I_i\}\ \{x_i \overset{l_{ij}}{\nrightarrow} \mid i \in J, j \in J_i\}}{f(\vec{x}) \xrightarrow{l} C[\vec{x},\vec{y}]}\right) =$$

if $k \in I$ then $\forall_{j \in J_k}\ \exists_{p',p'' \in T(\Sigma_P)}\ p_k \equiv l_{kj}.p' + p''$ and
if $k \in J$ then $\forall_{j \in J_k}\ \forall_{p',p'' \in T(\Sigma_P)}\ p_k \not\equiv l_{kj}.p' + p''$,
where $\equiv$ denotes equality up to E_{BCCSP}.

Intuitively, the axiom transforms $f(\vec{p})$ into a sum of closed terms covering all its execution possibilities. This is akin to Milner's well known expansion law for parallel composition of head normal forms. In order to obtain them we iterate through the set of f-defining rules and check if $\vec{p}$ satisfies their hypotheses by means of $\checkmark$. The predicate $\checkmark$ makes sure that, for a given rule, every component of $\vec{p}$ is a term with enough action prefixed summands satisfying all the positive premises that involve the component, and no summands prefixed with the actions from any of the corresponding negative premises.

Theorem 8.3.1. *Consider a TSS $\mathcal{T} = (\Sigma, L, \mathcal{R})$ that is semantically well founded in the sense of [6, Definition 5.1], such that* BCCSP $\sqsubseteq \mathcal{T}$. *$E_{\mathcal{T}}$ is sound and ground-complete for strong bisimilarity on $T(\Sigma_P)$.*

Proof. Soundness follows in the standard fashion. Every transition $f(\vec{p})$ can perform is matched by the right hand side of the equation and vice versa due to the natural derivation of the execution tree according to the defining rules.

As shown in [6], in order to prove ground-completeness of an axiom system, it is sufficient to show that it is head normalizing, which means that it can bring any closed term to a h.n.f. Note that the axiomatization presented in Definition 8.3.5 always derives terms in h.n.f. □

In order to generate a Maude equational theory for the operations in a specification we use the command `(derive axiom schemas SPECIFICATION .)`. Just like in the case of the simulator component, the command both prepares the environment to perform equational reductions according to the generated axioms, and outputs a Maude specification that can be used externally, independently

of the environment. The generated equational theory has the name of the specification with the suffix "`-SCHEMA`" and is selected using the command `(select SPECIFICATION-SCHEMA .)`.

The standard command `reduce` derives the normal form of a given closed term. For example, having loaded the specification of _||_ from Section 8.3.1, this is how we obtain the normal form of $a.\mathbf{0} \parallel b.\mathbf{0}$:

```
> (reduce a . 0 || b . 0 .)

result PClosedTerm :  a . b . 0 + b . a . 0
```

To illustrate what the axiomatizations look like, consider a general binary operation between labels *mix* and an operation g defined as:

$$\frac{x \xrightarrow{k} x' \;\; y \xrightarrow{l} y' \;\; x \overset{l}{\not\to} \;\; y \overset{k}{\not\to}}{g(x,y) \overset{mix(k,l)}{\to} x' + y'} \qquad \frac{x \xrightarrow{l} x' \;\; y \xrightarrow{l} y'}{g(x,y) \xrightarrow{l} \mathbf{0}}.$$

The first equation derives a sum of new operations, g_1 and g_2, one for each rule defining g. These new operations have the same domain as g, only extended with one parameter that will ultimately hold the tree of all execution paths that start with the corresponding rule – its head normal form. Initially this parameter is set to $\mathbf{0}$.

```
eq g(x,y) = g1(x,y,0) + g2(x,y,0) .
```

Let us first present the axiom for g_1, which is given as a standard Maude conditional equation, and then discuss all of its aspects.

```
ceq g1(x,y,SOLUTION) = g1(x,y,SOLUTION')  --- (0)
 if k . x' + x1 := x + dummy              --- (1)
 /\ l . y' + x2 := y + dummy              --- (2)
 /\ not(x can l)                          --- (3)
 /\ not(y can k)                          --- (4)
 /\ NEW-SUMMAND := mix(k, l).(x' + y')    --- (5)
 /\ SOLUTION' := SOLUTION + NEW-SUMMAND   --- (6)
 /\ SOLUTION =/= SOLUTION'                --- (7)
 .
```

Condition (1) requires that the first parameter satisfies the formula $x \xrightarrow{k} x'$. The variable x needs to be matched by a term that has $k.x'$ as a summand (x_1 is a generic variable of sort `PTerm`). If x is exactly of the shape $k.\_$ then Maude cannot find a match between $k.x' + x_1$ and x, not knowing that it can assign $\mathbf{0}$ to x_1, which explains the use of a constant of sort `PTerm` denoted by `dummy` added as a summand on the right hand side.

Condition (3) requires that x satisfies the formula $x \overset{l}{\nrightarrow}$. The natural inductive definition of the operation *can* is included in the module ET-BCCSP:

```
op _can_ :  PTerm PLabel -> Bool .

eq       0 can l = false .
eq (x + y) can l = (x can l) or (y can l) .
eq (l . x) can l = true .
ceq (l . x) can k = false if l =/= k .
```

If conditions (1) – (4) are satisfied, then the premises of g's first rule are met. This means that $mix(k, l).(x' + y')$, set at line (5) as the value for the variable NEW-SUMMAND, has to be a summand of the resulting head normal form. This head normal form is computed incrementally, by finding such summands individually, using the third parameter of g_1: SOLUTION and SOLUTION' hold the head normal forms computed before and, respectively after the current call of g_1. The aforementioned summand is added only if it is not already part of SOLUTION (conditions (6) – (7)). Should all conditions hold, a recursive call of g_1 is initiated (line (0)).

An important fact to keep in mind is that, in the specification, for any given rule, the labels of negative transitions given as variables need to also appear on some of the positive ones. For instance, had we not had the premise $x \overset{k}{\rightarrow} x'$, where k is a variable over the set of labels $L = \{a, b, c\}$, it would have been impossible to tell if condition (4) is met due to the missing assignment for k that should have resulted when evaluating condition (1). It is possible, however, to have a rule with negative premises labelled directly with constants, without the need for those constants to appear in other premises of the same rule.

If any of the conditions (1) – (4) does not hold, or if no new solution is found, then the following base-case equation is called:

```
eq g1(x,y,SOLUTION) = SOLUTION [owise] .
```

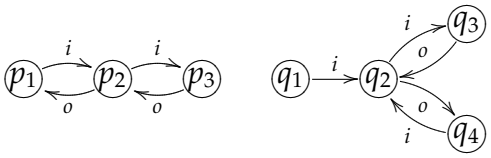
The equations for g_2 are generated in a similar fashion:

```
ceq g2(x,y,SOLUTION) = g2(x,y,SOLUTION')
 if l . x' + x1 := x + dummy
 /\ l . y' + x2 := y + dummy
 /\ NEW-SOLUTION := l . 0
 /\ SOLUTION' := SOLUTION + NEW-SOLUTION
 /\ SOLUTION =/= SOLUTION'
 .
 eq g2(x,y,SOLUTION) = SOLUTION [owise] .
```

In Meta SOS one can also specify recursive processes. If two such processes are given in BCCSP with guarded recursion in order to determine whether they are

bisimilar, the user can call a decision procedure implementing a unique fixed point induction algorithm. Currently the user needs to make sure that the guardedness condition is met.

By way of example, consider the following transition systems.



We specify this behaviour in Meta SOS by means of the reserved operation `def`:

```
ops p1 p2 p3 q1 q2 : -> PClosedTerm .  ops i o : -> PAction .

eq def(p1) = i . p2 .              eq def(q1) = i . q2 .
eq def(p2) = i . p3 + o . p1 .     eq def(q2) = i . q3 + o . q4 .
eq def(p3) = o . p2 .              eq def(q3) = o . q2 .
                                   eq def(q4) = i . q2 .
```

The command `(reduce areEqual(p1, q1) .)` checks whether p_1 and q_1 are bisimilar. The output in this case is the pair `< true ; < p1 ; q1 > < p1 ; q4 > < p2 ; q2 > < p3 ; q3 > >`, where the first element of the pair indicates whether the processes are bisimilar, and, if this is indeed the case, the second one is a representation of the found bisimulation.

8.3.3 Commutativity Format Checker

Besides automatically deriving sound and ground-complete axiomatizations, the focus of Meta SOS is also to check for algebraic properties of operations, by design. We have implemented a component that analyzes the provided SOS specification in order to find binary operations that are commutative. We adapt the format for binary operations from [117] to GSOS systems that may have negative premises.

Definition 8.3.6 (Commutativity)**.** *Given a TSS and a binary process operator f in its process signature, f is called* commutative w.r.t. a relation $\sim$*, if the following equation is sound w.r.t. $\sim$:*

$$f(x_0, x_1) = f(x_1, x_0).$$

Definition 8.3.7 (Commutativity format [117]). *A transition system specification over signature Σ is in* comm-form *format with respect to a set of binary function symbols $COMM \subseteq \Sigma$ if all its f-defining transition rules with $f \in COMM$ have the following form*

$$\textbf{(c)}\ \frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in \{0,1\}, j \in I_i\} \cup \{x_i \overset{l_{ij}}{\nrightarrow} \mid i \in \{0,1\}, j \in J_i\}}{f(x_0, x_1) \xrightarrow{l} t}$$

where I_i and J_i are finite index sets for each $i \in \{0,1\}$, and variables appearing in the source of the conclusion and target of the premises are all pairwise distinct. We denote the set of premises of **(c)** *by H. Moreover, for each such rule, there exist a transition rule* **(c')** *of the following form in the transition system specification*

$$\textbf{(c')}\ \frac{H'}{f(x'_0, x'_1) \xrightarrow{l} t'}$$

and a bijective mapping (substitution) $\hbar$ on variables such that (1) $\hbar(x'_0) = x_1$ and $\hbar(x'_1) = x_0$, (2) $\hbar(t') \sim_{cc} t$ and (3) $\hbar(h') \in H$, for each $h' \in H'$. Here $\sim_{cc}$ means equality up to swapping of arguments of operators in COMM in any context. Transition rule **(c')** *is called the* commutative mirror *of* **(c)**.

Theorem 8.3.2 (Commutativity for comm-form [117]). *If a transition system specification is in* comm-form *format with respect to a set of operators COMM, then all operators in COMM are commutative with respect to strong bisimilarity.*

We implement an algorithm that, for a given operation, searches for all of its rules that are commutative mirrors. It is well known that parallel composition is commutative. To check this using our tool we load the specification presented in Section 8.3.1 and call `(check formats SPECIFICATION .)`. The output shows that the first two rules defining _||_ are commutative mirrors, and that the third rule involving the termination predicate ↓ is a commutative mirror of itself, by pointing out the bijective mapping:

```
_||_ is commutative:

x -(alpha)-> x'                          y -(alpha)-> y'
===                          mirrors     ===
x || y -(alpha)-> x' || y                x || y -(alpha)-> x || y'

with:  alpha <- alpha  x' <- y'  x <- y  y' <- x'  y <- x

x -(|)-> x' , y -(|)-> y'                x -(|)-> x' , y -(|)-> y'
===                          mirrors     ===
x || y -(|)-> 0                          x || y -(|)-> 0
```

```
  with:  x' <- y'  x <- y  y' <- x'  y <- x
```

What Meta SOS does internally is to generate a Maude theory that has the name of the specification with the suffix "-FORMATS". It is the same as the initial specification, only that all the "sos" operators that are found commutative are enhanced with the attribute comm. This is of use both when having to perform rewrites modulo commutativity involving those operations, and as meta-information for future components that may need it. One of these components could, for instance, be dedicated to optimizing axiomatizations, using the approach presented in Chapter 5.

An important thing to remark is that the label mapping `alpha <- alpha` appears amongst the process variables mapping. The reason we extend the mapping to labels too is the fact that the user should not be forced to use the same variable name for matching premises of different rules. We would thus find that the first two rules are commutative mirrors even if they had different variables for actions, e.g. `alpha` and `beta`, respectively.

Aside from giving the user more freedom when choosing names for label variables, extending the mapping to labels is actually necessary for proving that some operators are commutative. Consider, for example, the operation g introduced in Section 8.3.2 and assume that the operation *mix* over labels is declared as commutative. Suppose label variables were not taken into account when searching for commutative mirrors. Then there would be no way of directly proving that g is commutative, unless the user specified the 6 instantiations of the first rule for g, involving the concrete action labels a, b, c.

```
> (check formats SPECIFICATION .)

g is commutative:

x-(k)->x',y-(l)->y',x-(l)/>,y-(k)/>
===
g(x,y) -(mix(k,l))-> x' + y'

mirrors

x-(k)->x',y-(l)->y',x-(l)/>,y-(k)/>
===
g(x,y) -(mix(k,l))-> x' + y'

with: k <- l  l <- k  x' <- y'  x <- y  y' <- x'  y <- x


x -(l)-> x' , y -(l)-> y'          x -(l)-> x' , y -(l)-> y'
===                      mirrors  ===
```

```
g(x,y) -(l)-> 0                    g(x,y) -(l)-> 0

with: l <- l  x' <- y'  x <- y  y' <- x'  y <- x
```

If we look at the first rule, note that when applying the substitution on labels, in order to check for the commutativity format, we need to make sure that `mix(k,l)` and `mix(l,k)` stand for the same label. This holds in our case because we do not merely check for syntactic equality, but for equality within the algebra defined for labels. Recall that we consider *mix* to be commutative.

The first rule is found as a mirror of itself based on the commutativity of _+_. Had the consequent of the rule been of the shape `x' * y'` (_*_ being a new binary operation), Meta SOS would have attempted to prove first that _*_ is commutative.

8.3.4 Linda – Integrating Components

In this section we present another case study and show how easy it is to make use of the functionality provided by all the previously described components. Let us focus on the tuple-space based coordination language Linda [55] and its SOS semantics, as given in [116].

Consider a minimalistic signature for the data component, Σ_D, that consists of constants for tuples (typical members u, v) and two operations for working with multisets of tuples: $\emptyset$ for the empty multiset and __ (blank) as a commutative and associative binary separator for the elements from the multiset. The operation __ has $\emptyset$ as identity element. We prefer to use constructs instead of the standard mathematical ones (braces "{", "}" for set separators, commas "," for separating elements within a set, and set union operator "$\cup$") for implementation purposes. For instance, the multiset $\{u, v\} \cup \{u\} \cup \emptyset$ is written as $u\ v\ u\ \emptyset$ in $\mathbb{T}(\Sigma_D)$, which is the same as $u\ v\ u$ because $\emptyset$ is the identity element. (That is actually the standard Maude notation for sets and multisets.) This is how we declare the above mentioned signature:

```
sort PData PClosedData .  subsort PClosedData < PData  .
op  empty : -> PData .
op     __ : PData PData -> PData [assoc comm id: empty] .
ops   u v : -> PClosedData .
```

Linda has several constructs for manipulating a shared data component of the language:

- *ask*(u) and *nask*(u) check, respectively, whether tuple u is (or is not) in the data space,
- *tell*(u) adds tuple u to the data space,
- *get*(u) removes tuple u from the data space.

The *ask*(u) and *get*(u) operations are blocking, in the sense that a process executing them blocks if u is not in the data space. *nask*(u) is also blocking if u is in the data space.

In Chapter 4 we show how to use labels for operating with the data component. For Linda, the set of labels L is extended to triples of the from $\langle d, -, d' \rangle$, where d, d' are open data terms from $\mathbb{T}(\Sigma_D)$, standing for the store before and, respectively, after the transition. The language does not have actions, hence the use of the placeholder "$-$" within the triple. As shown later, in order to have a finite set of labels and rules, which is necessary to have a proper GSOS system, we use symbolic names instead of open data terms.

Besides the four constructs for operating with the store, the language includes the prefix operation $l._-$ (for every l in L), nondeterministic choice $\_ + \_$, parallel composition $\_ \| \_$, and sequential composition $\_ ; \_$, all in the context of the already introduced termination predicate $\downarrow$. Linda also comes with a successfully terminated process, which we denote by $\downarrow .\mathbf{0}$.

In order to handle the store, our approach of extending the prefix operation to triples is slightly different from the one in Chapter 4. Though less intuitive, it is easier to implement than the one involving two new operations, *check* and *update*, because it requires no extra core axioms aside from those in E_{BCCSP}.

```
op <_,_,_> : PData PAction PData -> PLabel .  op - : -> PAction .
```

We first make sure that the SOS specification disjointly extends BCCSP, as required by Theorem 8.3.1. The rules for Σ_{BCCSP} are declared as presented in Section 8.2 because they are the same both for the extended labels and the termination predicate.

Given that μ is a variable to be replaced by any considered constant tuple, the rules for the operations manipulating the data component are:

$$\frac{}{ask(\mu) \overset{\langle x_D\ \mu, -, x_D\ \mu \rangle}{\rightarrow} \downarrow .\mathbf{0}} \qquad \frac{}{tell(\mu) \overset{\langle x_D, -, x_D\ \mu \rangle}{\rightarrow} \downarrow .\mathbf{0}} \qquad \frac{}{get(\mu) \overset{\langle x_D\ \mu, -, x_D \rangle}{\rightarrow} \downarrow .\mathbf{0}}.$$

Linda also has a basic operation named *nask* that checks if a tuple is not in the tuple space. The operation, however, is defined using side conditions, and currently we provide no support for such rules.

For the purpose of demonstration, we will only implement a limited and artificial version of Linda:

```
ops ask tell get : PClosedData -> PTerm [metadata "sos"] .
op  d : -> PData .
var mu : PClosedData .

===
tell(mu) -(<d, -, (d mu)>)-> |.0

===
ask(mu) -( <(d mu), -, (d mu)> )-> |.0

===
get(mu) -( <(d mu), -, d> )-> |.0
```

The limitation consists in the use of a symbolic constant `d`, denoting a data term, instead of a variable of the same sort. This is because in [77] it is presented how to derive a sound and ground-complete axiomatization modulo a notion of bisimilarity only for systems with a data component whose domain is a finite set of constants, and not a (possibly infinite) set of open terms. In our case the domain of the data component can be thought of as a set of constants, limited to the number of tuples taken into account plus one (for the symbolic constant).

Using the constant `d` is also useful during the axiomatization process because it helps avoiding generating equations with fresh variables on the right hand side. For instance, according to the schema from Definition 8.3.5, the following axiom $tell(\mu) = \langle d, -, d\ \mu\rangle.\downarrow.\mathbf{0}$ is generated, and here it is required that d is not a variable.

The rules for _||_ are very similar to those shown in Section 8.3.1, and those for _;_ are:

$$\frac{x \overset{\langle x_D,-,x'_D\rangle}{\longrightarrow} x'}{x\ ;\ y \overset{\langle x_D,-,x'_D\rangle}{\longrightarrow} x'\ ;\ y} \qquad \frac{x \xrightarrow{\downarrow} x' \quad y \overset{\langle x_D,-,x'_D\rangle}{\longrightarrow} y'}{x\ ;\ y \overset{\langle x_D,-,x'_D\rangle}{\longrightarrow} y'} \qquad \frac{x \xrightarrow{\downarrow} x' \quad y \xrightarrow{\downarrow} y'}{x\ ;\ y \xrightarrow{\downarrow} \mathbf{0}}\ .$$

The rules for the last two operations do not introduce new names for data terms on the consequent transitions (all the names are known from the premises), which means that no axioms with fresh variables on the right hand side can be generated.

Therefore it is safe to declare them using variables of sort `PData` instead of symbolic constants.

```
op _;_ : PTerm PTerm -> PTerm [metadata "sos"] .
var xD xD' : PData .

x -(<xD,-,xD'>)-> x'
===
x;y -(<xD,-,xD'>)-> (x';y)

x -(|)-> x', y -(<xD,-,xD'>)-> y'
===
x;y -(<xD,-,xD'>)-> y'

x -(|)-> x', y -(|)-> y'
===
x;y -(|)-> y'
```

A use case scenario involving all the components illustrated so far may start with loading the specification for Linda and checking which operations are commutative (`check formats LINDA .`). Remark that `_;_`'s commutativity cannot be proven:

```
Could not prove commutativity for:  _;_
Could not find commutative mirrors within:
x -(<xD,-,xD'>)-> x'
===
x;y -(<xD,-,xD'>)-> (x';y)

x -(|)-> x', y -(<xD,-,xD'>)-> y'
===
x;y -(<xD,-,xD'>)-> y'

x -(|)-> x', y -(|)-> y'
===
x;y -(|)-> y'
```

We could continue by deriving the axiom schema and determining the normal form of a term such as *ask*(*u*) ; *tell*(*v*). Finally we can check if indeed the found normal form is bisimilar to the initial term.

```
> (derive axiom schemas LINDA-FORMATS .)
> (select LINDA-FORMATS-SCHEMA .)
> (reduce ask(u) ; tell(v) .)
result PClosedTerm :  < d u,-,d u > . < d,-,d v > . | . 0
> (derive simulator LINDA-FORMATS .)
> (check (ask(u) ; tell(v)) ~ (< d u,-,d u > . < d,-,d v > . | . 0) .)
result: true.Bool
```

8.3.5 Adding Components

Meta SOS is conceived in a way to be easily extended with new components. Besides the three components presented in Sections 8.3.1, 8.3.2 and 8.3.3 the tool includes a file named `component-sample.maude` which the user can adapt to implement a new desired functionality by following the patterns presented in [83].

In what follows, the name "sample" is generic and is meant to be replaced by some other name suggesting the functionality of a new component. Each component has two modules `SAMPLE-LANG-SIGN` and `SAMPLE-STATE-HANDLING`, dedicated for the signature of the implemented commands and, respectively, their semantics. Once implemented, the functionality is included in the Meta SOS framework by following these steps: (1) in the file `metasos-interface.maude` the signature for the new commands needs to be included in the `METASOS-LANG-SIGN` module and their semantics needs to be included in the module `METASOS-STATE-HANDLING`, (2) in the file `metasos.maude` the new component needs to be loaded just like the others: `load component-sample.maude`.

It is worth mentioning that, in order to ease the development cycle, the framework provides support for unit testing. It is beyond the scope of this chapter, though, to present how to make use of this facility.

8.4 Conclusion and Future Work

Meta SOS addresses many of the extensions foreseen in [114]. Namely, it represents a core framework dedicated to implementing SOS meta-theorems, it provides support for generating axiomatizations, and it frees the user from implementing matching procedures for specified language constructs. In its present form, Meta SOS can handle languages whose operational specification is in the GSOS format, such as most classic process calculi and Linda. Another aspect addressed in [114] is the support for more general SOS frameworks that allow for terms as labels, as well as multi-sorted and binding signatures. This would allow the framework to handle name-passing and higher-order languages such as the π-calculus [129]. Though Meta SOS does not provide this kind of support yet, the general way in which it handles labels is a good step towards that goal.

There are, naturally, many ways to improve and extend the tool. Besides checking for the commutativity format, there are many other formats to check for: deter-

minism and idempotence [5] (see also Chapter 6), zero and unit elements [10], associativity [61], and distributivity [9]. Adapting PREG Axiomatizer and adding it as a component to Meta SOS as presented in Section 8.3.5 would also be of value due to its different approach to generating axiomatizations, and because it includes a GSOS format checker. The axiomatization process could be enhanced using the technique presented in Chapter 5. This would lead to smaller and more natural axiom systems.

Chapter 9

Conclusions and Future Work

The theory of SOS has been evolving for over three decades, serving as a natural way of providing and analyzing language semantics. The purpose of this work was to continue this evolution by considering SOS rules with predicates and data, and analyzing these rules in order to derive axiomatizations and properties of language constructs.

In Chapter 2 we proposed a sound and ground-complete axiomatization modulo stateless bisimilarity for Linda. This served as an introductory case study of a language that has a data component, as well as a predicate. It also highlighted the power of equational reasoning and the structure of ground-completeness proofs in a concrete setting.

Chapters 3 and 4 then presented the context of extending GSOS systems with arbitrary predicates and, respectively, data components, and how to automatically generate ground-complete axiomatizations modulo strong and, respectively, stateless bisimilarity for these systems. The work on predicates also includes a thorough analysis on different types of predicates that can be specified, depending on process execution paths. While considering systems with the store component we also present how to lift already existing rule formats for algebraic properties to this setting.

In Chapter 5 we presented an extension of the format for automatically identifying operations with commutative arguments, and showed how to use it in order to obtain smaller and more natural axiomatizations of bisimilarity over GSOS languages.

Chapter 6 gave a contribution to the meta-theory of SOS rule formats for idempotence, and consists of a rule format for automatically identifying unary and binary idempotent terms.

Chapters 7 and 8 represent the practical aspect of the current thesis. One presents a tool for deriving axiomatizations for GSOS with predicates, while the other is a core framework dedicated to integrating results from the meta-theory of SOS. The framework currently includes modules for deriving axiomatizations for basic GSOS specifications, performing simulations and identifying binary operators that comply to a commutativity rule format.

Naturally, there are many ways to continue the development of the meta-theory of SOS. We point out some important lines for future work.

- **Axiomatizations for other notions of behavioural equivalence.** Strong bisimilarity is an instructive, but, in many cases, too a restrictive behavioural equivalence relation. *Weak bisimilarity*, for instance, is a version of strong bisimilarity that ignores silent moves and has more practical value. A viable way of conceiving axiomatizations modulo weak bisimilarity is to begin with formats for this notion of equivalence which guarantee that it is a congruence, some of which are presented in [47, 80], and build upon the results in [136], where an axiomatization for testing a preorder for arbitrary De Simone languages is proposed.

- **Considering other rule formats for optimizing axiomatizations.** We have shown that for commutative operations it is now easier to generate natural axiomatizations that are close to those that already exist in the literature. The question is whether other properties such as associativity or distributivity can be used for the same purpose.

- **Rule formats derived from laws.** Much effort has been put into developing rule formats for guaranteeing basic properties. It would be of interest to investigate to which extent, for a given algebraic law, rule formats that guarantee the law can automatically derived.

- **Meta-theory of Nominal SOS.** Many languages support concepts such as variables, name abstraction (binding), or facilities for the recursive definition of processes. A line of research of considerable interest would be to develop a meta-theory of SOS with these aspects, named *nominal aspects* [71, 72, 121], building on [58, 67, 73, 97, 131]. So far there have been efforts to integrate these in SOS meta-theories, which materialized in frameworks

for establishing sufficient syntactic conditions guaranteeing the validity of a semantic result (congruence in the case of [45, 67, 102, 140] and conservativity in the case of [103, 70]). However there are many aspects to be handled before the meta-theory of Nominal SOS can reach that of classic SOS. Deriving, for instance, a Nominal GSOS format that facilitates automatically obtaining sound and ground-complete axiomatizations for suitable notions of bisimilarity, as well as lifting already existing rule formats for algebraic properties would be of great interest.

- **Extending Meta SOS.** Though the current features alone make Meta SOS a useful tool, they represent but a fraction of the potential that the meta-theory of SOS has. Checking for systems conformity to already existing rule formats for pointing out the congruence of behavioural equivalences, algebraic properties, or conservative extensions would be of value to language designers and could be implemented within the framework by means of additional modules. Optimizing axiomatizations and analyzing their performance is also desirable.

Bibliography

[1] Luca Aceto (1993): *Deriving Complete Inference Systems for a Class of GSOS Languages Generating Regular Behaviours*. Report IR 93–2009, Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg University, Aalborg. Also available as Computer Science Report 1/94, University of Sussex, January 1994. [cited at p. 119]

[2] Luca Aceto (1994): *Deriving Complete Inference Systems for a Class of GSOS Languages Generating Regular Behaviours*. In Bengt Jonsson & Joachim Parrow, editors: *Proceedings of the fifth International Conference on Concurrency Theory (CONCUR'94), Lecture Notes in Computer Science* 836, Springer-Verlag, Berlin, Germany, pp. 449–464, doi:10.1007/BFb0015025. [cited at p. 7, 32, 52, 53, 103, 106, 134]

[3] Luca Aceto (1994): *GSOS and finite labelled transition systems*. *Theoretical Computer Science* 131, pp. 181–195. [cited at p. 106, 134]

[4] Luca Aceto (2007): *Reactive systems: modelling, specification and verification*. Cambridge University Press. Available at `http://books.google.com/books?id=Ju0HM-2RIwgC`. [cited at p. 1]

[5] Luca Aceto, Arnar Birgisson, Anna Ingólfsdóttir, Mohammad Reza Mousavi & Michel A. Reniers (2012): *Rule formats for determinism and idempotence*. *Science of Computer Programming* 77(7–8), pp. 889–907, doi:10.1016/j.scico.2010.04.002. [cited at p. 79, 95, 102, 132, 136, 140, 151, 167, 187]

[6] Luca Aceto, Bard Bloom & Frits Vaandrager (1994): *Turning SOS rules into equations*. *Inf. Comput.* 111, pp. 1–52, doi:10.1006/inco.1994.1040. [cited at p. 7, 31, 32, 33, 38, 42, 43, 45, 48, 49, 50, 52, 53, 59, 60, 61, 62, 79, 84, 86, 87, 91, 103, 104, 106, 108, 115, 116, 117, 119, 120, 122, 123, 124, 125, 126, 134, 159, 161, 165, 175, 176]

[7] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac & Anna Ingólfsdóttir (2011): *Axiomatizing GSOS with Predicates*. In Michel A. Reniers & Pawel

Sobocinski, editors: *SOS*, *EPTCS* 62, pp. 1–15, doi:10.4204/EPTCS.62.1. [cited at p. 9, 33]

[8] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac & Anna Ingólfsdóttir (2011): *PREG Axiomatizer – A Ground Bisimilarity Checker for GSOS with Predicates*. In: *CALCO'11*, pp. 378–385, doi:10.1007/978-3-642-22944-2_27. [cited at p. 10]

[9] Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Mohammad Reza Mousavi & Michel A. Reniers (2011): *Rule Formats for Distributivity*. In Adrian Horia Dediu, Shunsuke Inenaga & Carlos Martín-Vide, editors: *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26–31, 2011. Proceedings*, *Lecture Notes in Computer Science* 6638, Springer, pp. 80–91, doi:10.1007/978-3-642-21254-3_5. [cited at p. 79, 95, 102, 132, 136, 187]

[10] Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Mohammad Reza Mousavi & Michel A. Reniers (2011): *SOS rule formats for zero and unit elements*. *Theoretical Computer Science* 412(28), pp. 3045–3071, doi:10.1016/j.tcs.2011.01.024. [cited at p. 79, 95, 102, 132, 136, 162, 187]

[11] Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir & Bas Luttik (2005): *Finite Equational Bases in Process Algebra: Results and Open Questions*. In: *Processes, Terms and Cycles*, *Lecture Notes in Computer Science* 3838, Springer, pp. 338–367, doi:10.1007/11601548_18. [cited at p. 104, 126]

[12] Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir & Bas Luttik (2009): *A finite equational base for CCS with left merge and communication merge*. *ACM Trans. Comput. Log.* 10(1), doi:10.1145/1459010.1459016. [cited at p. 126]

[13] Luca Aceto, Willem Jan (Wan) Fokkink & Chris Verhoef (2001): *Conservative Extension in Structural Operational Semantics*. In Gheorghe Paun, Grzegorz Rozenberg & Arto Salomaa, editors: *Current Trends in Theoretical Computer Science - Entering the 21st Century*, World Scientific, Singapore, pp. 504–524, doi:10.1007/11523468_98. [cited at p. 6]

[14] Luca Aceto, Willem Jan (Wan) Fokkink & Chris Verhoef (2001): *Structural Operational Semantics*. In Jan A. Bergstra, Alban Ponse & Scott A. Smolka, editors: *Handbook of Process Algebra*, Elsevier Science, Dordrecht, The Netherlands, 2001, pp. 197–292, doi:10.1016/B978-044482830-9/50021-7. [cited at p. 6, 7, 105, 106, 133, 134, 135, 167, 171]

[15] Luca Aceto, Eugen-Ioan Goriac & Anna Ingólfsdóttir (2013): *A Ground-Complete Axiomatization of Stateless Bisimilarity over Linda*. Technical Report, Reykjavik University. Available at http://www.ru.is/faculty/luca/PAPERS/axiomatizing_linda.pdf. [cited at p. 9]

[16] Luca Aceto, Eugen-Ioan Goriac & Anna Ingólfsdóttir (2013): *Meta SOS – A Maude Based SOS Meta-Theory Framework*. In: *Proceedings Combined 20th International Workshop on Expressiveness in Concurrency and 10th Workshop on Structural Operational Semantics*, Lecture Notes in Computer Science. To appear. [cited at p. 11]

[17] Luca Aceto, Eugen-Ioan Goriac & Anna Ingólfsdóttir (2013): *SOS Rule Formats for Idempotent Terms and Idempotent Unary Operators*. In P. van Emde Boas et al., editor: *Proceedings of SOFSEM 2013, Lecture Notes in Computer Science* 7741, Springer-Verlag, pp. 108–120. [cited at p. 10, 133]

[18] Luca Aceto, Eugen-Ioan Goriac, Anna Ingólfsdóttir, Mohammad Reza Mousavi & Michel Reniers (2013): *Exploiting Algebraic Laws to Improve Mechanized Axiomatizations*. In: *Proceedings of the 5th Conference on Algebra and Coalgebra in Computer Science (CALCO 2013), Lecture Notes in Computer Science* 8089, Springer-Verlag, Berlin, Germany, 2013. [cited at p. 10, 105]

[19] Luca Aceto & Matthew Hennessy (1992): *Termination, Deadlock, and Divergence*. *Journal of the ACM* 39, pp. 147–187, doi:10.1145/147508.147527. [cited at p. 7]

[20] Luca Aceto & Anna Ingolfsdottir (1996): *CPO Models for Compact GSOS Languages*. *Information and Computation* 129(2), pp. 107–141, doi:10.1006/inco.1996.0077. [cited at p. 106, 134]

[21] Luca Aceto, Anna Ingolfsdottir & Eugen-Ioan Goriac (2014): *SOS Rule Formats for Idempotent Terms and Idempotent Unary Operators*. *The Journal of Logic and Algebraic Programming* 83(1), pp. 64 – 80, doi:10.1016/j.jlap.2013.07.003. [cited at p. 10]

[22] Luca Aceto, Anna Ingólfsdóttir, Bas Luttik & Paul van Tilburg (2008): *Finite Equational Bases for Fragments of CCS with Restriction and Relabelling*. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri & C.-H. Luke Ong, editors: *Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of*

Computer Science, September 7–10, 2008, Milano, Italy, *IFIP* 273, Springer, pp. 317–332, doi:10.1007/978-0-387-09680-3_22. [cited at p. 126]

[23] Luca Aceto, Anna Ingólfsdóttir, MohammadReza Mousavi & Michel A. Reniers (2009): *Algebraic Properties for Free!* *Bulletin of the European Association for Theoretical Computer Science (BEATCS)* 99, pp. 81–104. [cited at p. 84, 103, 125, 132, 136, 138, 151, 167]

[24] Franz Baader & Tobias Nipkow (1999): *Term Rewriting and All That*. Cambridge University Press. [cited at p. 79]

[25] J. C. M. Baeten, T. Basten & M. A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. *Cambridge Tracts in Theoretical Computer Science* 50, Cambridge University Press, Cambridge. [cited at p. 1, 3, 16, 18, 79, 92, 117, 119, 124, 131, 148, 163]

[26] J. C. M. Baeten & J. A. Bergstra (1988): *Global renaming operators in concrete process algebra*. *Information and Computation* 78(3), pp. 205–245, doi:10.1016/0890-5401(88)90027-2. [cited at p. 150]

[27] J. C. M. Baeten, J. A. Bergstra & J. W. Klop (1986): *Syntax and defining equations for an interrupt mechanism in process algebra*. *Fundamenta Informaticae* IX(2), pp. 127–168. [cited at p. 132, 149]

[28] J. C. M. Baeten & W. P. Weijland (1990): *Process Algebra*. Cambridge University Press, New York, NY, USA. [cited at p. 31, 48, 164]

[29] J.C.M. Baeten & Jan A. Bergstra (1996): *Discrete time process algebra*. *Formal Aspects of Computing* 8(2), pp. 188–208, doi:10.1007/BF01214556. [cited at p. 114]

[30] J.C.M. Baeten & Chris Verhoef (1993): *A Congruence Theorem for Structured Operational Semantics with Predicates*. In Eike Best, editor: *International Conference on Concurrency Theory (CONCUR'93)*, *Lecture Notes in Computer Science* 715, Springer-Verlag, Berlin, Germany, pp. 477–492, doi:10.1007/3-540-57208-2_33. [cited at p. 7, 38]

[31] J.C.M. Baeten, editor (1990): *Applications of Process Algebra*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press. Available at `http://books.google.is/books?id=dPmyAAAAIAAJ`. [cited at p. 1]

[32] Jos C. M. Baeten (2003): *Embedding untimed into timed process algebra: the case for explicit termination*. *Mathematical Structures in Computer Science* 13(4), pp. 589–618, doi:10.1017/S0960129503004006. [cited at p. 24]

[33] Jos C. M. Baeten (2005): *A brief history of process algebra*. *Theor. Comput. Sci.* 335(2-3), pp. 131–146, doi:10.1016/j.tcs.2004.07.036. [cited at p. 1, 3]

[34] Jos C. M. Baeten & Jan A. Bergstra (1997): *Process Algebra with Propositional Signals*. *Theoretical Computer Science (TCS)* 177(2), pp. 381–405, doi:10.1016/S0304-3975(96)00253-8. [cited at p. 80]

[35] Jos C. M. Baeten & Erik P. de Vink (2004): *Axiomatizing GSOS with termination*. *J. Log. Algebr. Program.* 60-61, pp. 323–351, doi:10.1016/j.jlap.2004.03.001. [cited at p. 7, 32, 103, 104, 106, 125, 134, 159, 161, 175]

[36] Christel Baier & Joost-Pieter Katoen (2008): *Principles of Model Checking*. MIT Press. [cited at p. 79]

[37] Falk Bartels (2002): *GSOS for Probabilistic Transition Systems*. In: *Proceedings of the 5th International Workshop on Coalgebraic Methods in Computer Science (CMCS'02), Electronic Notes in Theoretical Computer Science* 65, pp. 29–53, doi:10.1016/S1571-0661(04)80358-X. [cited at p. 6]

[38] D. A. van Beek, Ka Lok Man, Michel A. Reniers, J. E. Rooda & Ramon R. H. Schiffelers (2006): *Syntax and consistent equation semantics of hybrid Chi*. *J. Log. Algebr. Program.* 68(1-2), pp. 129–210, doi:10.1016/j.jlap.2005.10.005. [cited at p. 80, 83]

[39] D. A. van Beek, Michel A. Reniers, Ramon R. H. Schiffelers & J. E. Rooda (2007): *Foundations of a Compositional Interchange Format for Hybrid Systems*. In Alberto Bemporad, Antonio Bicchi & Giorgio C. Buttazzo, editors: *Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC'07), Lecture Notes in Computer Science* 4416, Springer, pp. 587–600, doi:10.1007/978-3-540-71493-4_45. [cited at p. 80, 83]

[40] J. A. Bergstra & J. W. Klop (1985): *Algebra of communicating processes with abstraction*. *Theoretical Computer Science* 37(1), pp. 77–121, doi:10.1016/0304-3975(85)90088-X. [cited at p. 143]

[41] J A Bergstra & J W Klop (1986): *Verification of an alternating bit protocol by means of process algebra*. In: *Proceedings of the International Spring School on Mathematical method of specification and synthesis of software systems '85*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 9–23, doi:10.1007/3-540-16444-8_1. [cited at p. 48]

[42] J. A. Bergstra & C. A. Middelburg (2007): *Synchronous cooperation for explicit multi-threading*. *Acta Informatica* 44, pp. 525–569, doi:10.1007/s00236-007-0057-9. [cited at p. 80, 83]

[43] Jan A. Bergstra & J. W. Klop (1982): *Fixedpoint semantics in process algebra*. Technical Report IW 206/82, Center for Mathematics, Amsterdam, The Netherlands. [cited at p. 28, 117, 123]

[44] Jan A. Bergstra & J. W. Klop (1984): *Process algebra for synchronous communication*. *Information and Control* 60(1-3), pp. 109–137. [cited at p. 114, 124, 137]

[45] Karen L. Bernstein (1998): *A congruence theorem for structured operational semantics of higher-order languages*. In: *Proceedings of the 13th IEEE Symposium on Logic In Computer Science (LICS'98)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 153–164, doi:10.1109/LICS.1998.705652. [cited at p. 191]

[46] Bard Bloom (1989): *Ready Simulation, Bisimulation, and the Semantics of CCS-like Languages*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. [cited at p. 104, 106, 107, 109, 115, 134, 135, 136, 151]

[47] Bard Bloom (1995): *Structural Operational Semantics for Weak Bisimulations*. *Theoretical Computer Science (TCS)* 146, pp. 25–68, doi:10.1016/0304-3975(94)00152-9. [cited at p. 6, 103, 190]

[48] Bard Bloom, Willem Jan (Wan) Fokkink & Robert Jan (Rob) van Glabbeek (2004): *Precongruence formats for decorated trace semantics*. *ACM Transactions on Computational Logic* 5(1), pp. 26–78, doi:10.1145/963927.963929. [cited at p. 6]

[49] Bard Bloom, Sorin Istrail & Albert R. Meyer (1995): *Bisimulation can't be traced*. *J. ACM* 42, pp. 232–268, doi:10.1145/200836.200876. [cited at p. 6, 7, 32, 33, 80, 85, 104, 106, 107, 115, 134, 135, 151, 159, 161, 171, 172, 174]

[50] Roland Bol & Jan Friso Groote (1996): *The Meaning of Negative Premises in Transition System Specifications*. *Journal of the ACM (JACM)* 43(5), pp. 863–914, doi:10.1145/234752.234756. [cited at p. 6]

[51] Marcello M. Bonsangue, Jan J. M. M. Rutten & Alexandra Silva (2009): *An Algebra for Kripke Polynomial Coalgebras*. In: *LICS*, IEEE Computer Society, pp. 49–58, doi:10.1109/LICS.2009.18. [cited at p. 53]

[52] Victor Bos & Jeroen J.T. Kleijn (2003): *Redesign of a Systems Engineering Language — Formalisation of χ*. *Formal Aspects of Computing* 15(4), pp. 370–389, doi:10.1007/s00165-003-0017-2. [cited at p. 13, 14, 17]

[53] D. J. B. Bosscher (1994): *Term Rewriting Properties of SOS Axiomatisations*. In: *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, TACS '94, Springer-Verlag, London, UK, pp. 425–439, doi:10.1007/3-540-57887-0_108. [cited at p. 117, 161]

[54] Ed Brinksma (1985): *A Tutorial on* Lotos. In Michel Diaz, editor: *Proc. Protocol Specification, Testing and Verification V*, North-Holland, Amsterdam, Netherlands, pp. 171–194. [cited at p. 8, 168]

[55] Antonio Brogi & Jean-Marie Jacquet (1998): *On the Expressiveness of Linda-like Concurrent Languages*. *Electr. Notes Theor. Comput. Sci.* 16(2), pp. 75–96, doi:10.1016/S1571-0661(04)00118-5. [cited at p. 13, 15, 16, 29, 92, 182]

[56] Nicholas Carriero & David Gelernter (1989): *Linda in Context*. *Communications of the ACM* 32(4), pp. 444–458, doi:10.1145/63334.63337. [cited at p. 9, 13, 15, 81]

[57] Fabricio Chalub & Christiano Braga (2007): *Maude MSOS Tool*. *Electron. Notes Theor. Comput. Sci.* 176, pp. 133–146, doi:10.1016/j.entcs.2007.06.012. [cited at p. 8, 161, 168, 169, 173]

[58] Matteo Cimini, Mohammad Reza Mousavi, Michel A. Reniers & Murdoch James Gabbay (2012): *Nominal SOS*. *Electr. Notes Theor. Comput. Sci.* 286, pp. 103–116, doi:10.1016/j.entcs.2012.08.008. [cited at p. 190]

[59] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn L. Talcott, editors (2007): *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. *Lecture Notes in Computer Science* 4350, Springer, doi:10.1007/978-3-540-71999-1_1. [cited at p. 8, 10, 11, 32, 125, 161, 168, 169]

[60] Robert J. Colvin & Ian J. Hayes (2011): *Structural Operational Semantics through Context-Dependent Behaviour*. *Journal of Logic and Algebraic Programming* 80(7), pp. 392–426, doi:10.1016/j.jlap.2011.05.001. [cited at p. 80]

[61] Sjoerd Cranen, Mohammad Reza Mousavi & Michel A. Reniers (2008): *A Rule Format for Associativity*. In Franck van Breugel & Marsha Chechik, editors: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, *Lecture Notes in Computer Science* 5201, Springer-

Verlag, pp. 447–461, doi:10.1007/978-3-540-85361-9_35. [cited at p. 50, 79, 94, 95, 102, 132, 133, 136, 138, 139, 142, 145, 151, 187]

[62] Pieter J.L. Cuijpers & Michel A. Reniers (2005): *Hybrid Process Algebra*. *Journal of Logic and Algebraic Programming* 62(2), pp. 191–245, doi:10.1016/j.jlap.2004.02.001. [cited at p. 13, 14, 17, 98, 101]

[63] Rocco De Nicola & Matthew Hennessy (1984): *Testing Equivalences for Processes*. *Theoretical Computer Science* 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0. [cited at p. 13]

[64] Rocco De Nicola & Rosario Pugliese (2000): *Linda-based applicative and imperative process algebras*. *Theoretical Computer Science* 238(1–2), pp. 389–437, doi:10.1016/S0304-3975(99)00339-4. [cited at p. 13]

[65] Pierpaolo Degano & Corrado Priami (2001): *Enhanced operational semantics*. *ACM Computing Surveys* 33(2), pp. 135–176, doi:10.1145/384192.384194. [cited at p. 80]

[66] Joost Engelfriet & Tjalling Gelsema (1999): *Multisets and Structural Congruence of the pi-Calculus with Replication*. *Theoretical Computer Science* 211(1–2), pp. 311–337, doi:10.1016/S0304-3975(97)00179-5. [cited at p. 137]

[67] Marcelo P. Fiore & Sam Staton (2009): *A congruence rule format for name-passing process calculi*. *Inf. Comput.* 207(2), pp. 209–236, doi:10.1016/j.ic.2007.12.005. [cited at p. 190, 191]

[68] Wan Fokkink (1994): *A complete equational axiomatization for prefix iteration*. *Information Processing Letters* 52(6), pp. 333–337, doi:10.1016/0020-0190(94)00163-4. [cited at p. 143]

[69] Wan Fokkink (1997): *Axiomatizations for the Perpetual Loop in Process Algebra*. In Pierpaolo Degano, Roberto Gorrieri & Alberto Marchetti-Spaccamela, editors: *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7–11 July 1997, Proceedings*, *Lecture Notes in Computer Science* 1256, Springer, pp. 571–581, doi:10.1007/3-540-63165-8_212. [cited at p. 142]

[70] Willem Jan (Wan) Fokkink & Chris Verhoef (1998): *A Conservative Look at Operational Semantics with Variable Binding*. *Information and Computation (I&C)* 146(1), pp. 24–54, doi:10.1006/inco.1998.2729. [cited at p. 6, 108, 191]

[71] Murdoch J. Gabbay & Andrew M. Pitts (2002): *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects of Computing (FAC)* 13(3–5), pp. 341–363, doi:10.1007/s001650200016. [cited at p. 190]

[72] Murdoch James Gabbay & Aad Mathijssen (2008): *Capture-avoiding substitution as a nominal algebra*. *Formal Asp. Comput.* 20(4-5), pp. 451–479, doi:10.1007/s00165-007-0056-1. [cited at p. 7, 190]

[73] Andrew Gacek, Dale Miller & Gopalan Nadathur (2009): *Reasoning in Abella about Structural Operational Semantics Specifications*. *Electr. Notes Theor. Comput. Sci.* 228, pp. 85–100, doi:10.1016/j.entcs.2008.12.118. [cited at p. 190]

[74] Fabio Gadducci & Ugo Montanari (2000): *The Tile Model*. In Gordon D. Plotkin, Colin Stirling & Mads Tofte, editors: *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, Boston, MA, USA, 2000, pp. 133–166. [cited at p. 80]

[75] Vashti Galpin, Luca Bortolussi & Jane Hillston (2013): *HYPE: Hybrid modelling by composition of flows*. *Formal Asp. Comput.* 25(4), pp. 503–541, doi:10.1007/s00165-011-0189-0. [cited at p. 80, 83]

[76] M. Gazda & W.J. Fokkink (2010): *Turning GSOS into equations for linear time-branching time semantics*. *2nd Young Researchers Workshop on Concurrency Theory - YR-CONCUR'10, Paris*. Available at `http://www.cs.vu.nl/~wanf/pubs/gsos.pdf`. [cited at p. 32]

[77] Daniel Gebler, Eugen-Ioan Goriac & Mohammad Reza Mousavi (2013): *Algebraic Meta-Theory of Processes with Data*. In: *Proceedings Combined 20th International Workshop on Expressiveness in Concurrency and 10th Workshop on Structural Operational Semantics*, Lecture Notes in Computer Science. To appear. [cited at p. 9, 184]

[78] David Gelernter (1985): *Generative Communication in Linda*. *ACM Transactions on Programming Languages and Systems* 7(1), pp. 80–112, doi:10.1145/2363.2433. [cited at p. 9, 13, 15]

[79] R.J. van Glabbeek (2001): *The Linear Time - Branching Time Spectrum I. The Semantics of Concrete, Sequential Processes*. In A. Ponse S.A. Smolka J.A. Bergstra, editor: *Handbook of Process Algebra*, Elsevier, pp. 3–99, doi:10.1007/3-540-57208-2_6. [cited at p. 31, 39, 85, 170]

[80] Rob Van Glabbeek (2005): *On cool congruence formats for weak bisimulations (extended abstract)*. In: *Proceedings of the 2nd International Colloquium on*

Theoretical Aspects of Computing (ICTAC'05), *Lecture Notes in Computer Science* 3722, Springer, pp. 318–333, doi:10.1007/11560647_21. [cited at p. 6, 103, 190]

[81] Robert Jan (Rob) van Glabbeek (2001): *The Linear Time - Branching Time Spectrum I*. In Jan A. Bergstra, Alban Ponse & Scott A. Smolka, editors: *Handbook of Process Algebra, Chapter 1*, Elsevier Science, Dordrecht, The Netherlands, 2001, pp. 3–100. [cited at p. 4, 107, 134]

[82] Robert Jan (Rob) van Glabbeek (2004): *The Meaning of Negative Premises in Transition System Specifications II*. *Journal of Logic and Algebraic Programming (JLAP)* 60-61, pp. 229–258, doi:10.1007/3-540-61440-0_154. [cited at p. 6, 107]

[83] Eugen-Ioan Goriac, Georgiana Caltais, Dorel Lucanu, Oana Andrei & Gheorghe Grigoras (2009): *Patterns for Maude Metalanguage Applications*. *Electr. Notes Theor. Comput. Sci.* 238(3), pp. 121–138, doi:10.1016/j.entcs.2009.05.016. [cited at p. 169, 186]

[84] Jan Friso Groote (1993): *Transition system specifications with negative premises*. *Theoretical Computer Science (TCS)* 118(2), pp. 263–299, doi:10.1016/0304-3975(93)90111-6. [cited at p. 6, 7]

[85] Jan Friso Groote & Alban Ponse (1994): *Process Algebra with Guards: Combining Hoare Logic with Process Algebra*. *Formal Aspects of Computing* 6(2), pp. 115–164, doi:10.1007/BF01221097. [cited at p. 13, 14, 17]

[86] Jan Friso Groote & Frits W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation As a Congruence*. *Information and Computation* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6. [cited at p. 4, 6, 7, 8, 80, 168]

[87] Pieter H. Hartel (1999): *LETOS - a lightweight execution tool for operational semantics*. *Software: Practice and Experience* 29(15), pp. 1379–1416, doi:10.1002/(SICI)1097-024X(19991225)29:15%3C1379::AID-SPE286%3E3.0.CO;2-V. [cited at p. 8, 168]

[88] M. Hennessy (1981): *A term model for synchronous processes*. *Information and Control* 51(1), pp. 58–75. [cited at p. 132, 143]

[89] Matthew Hennessy & Anna Ingólfsdóttir (1993): *Communicating Processes with Value-passing and Assignments*. *Formal Aspects of Computing* 5(5), pp. 432–466, doi:10.1007/BF01212486. [cited at p. 13]

[90] Matthew Hennessy & Anna Ingólfsdóttir (1993): *A Theory of Communicating Processes with Value Passing*. *Information and Computation* 107(2), pp. 202–236, doi:10.1006/inco.1993.1067. [cited at p. 13]

[91] Matthew Hennessy, Huimin Lin & Julian Rathke (2001): *Unique fixpoint induction for message-passing process calculi*. *Science of Computer Programming* 41(3), pp. 241–275, doi:10.1016/S0167-6423(01)00008-9. [cited at p. 13]

[92] Matthew Hennessy & Robin Milner (1985): *Algebraic laws for nondeterminism and concurrency*. *J. ACM* 32(1), pp. 137–161, doi:10.1145/2455.2460. [cited at p. 116, 175]

[93] Matthew Hennessy & Gordon D. Plotkin (1979): *Full Abstraction for a Simple Parallel Programming Language*. In Jirí Becvár, editor: *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3–7, 1979*, *Lecture Notes in Computer Science* 74, Springer, pp. 108–120, doi:10.1007/3-540-09526-8_8. [cited at p. 3]

[94] Matthew Hennessy & Gordon D. Plotkin (1979): *Full Abstraction for a Simple Parallel Programming Language*. In Jirí Becvár, editor: *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science (MFCS'79)*, *Lecture Notes in Computer Science* 74, Springer-Verlag, Berlin, Germany, 1979, pp. 108–120, doi:10.1007/3-540-09526-8_8. [cited at p. 103]

[95] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs. [cited at p. 1, 3, 8, 31, 114, 131, 137, 143, 168]

[96] S.C. Kleene (1956): *Representation of events in nerve nets and finite automata*. In C.E. Shannon & J. McCarthy, editors: *Automata Studies*, Princeton University Press, pp. 3–41. [cited at p. 132, 141]

[97] Matthew R. Lakin & Andrew M. Pitts (2007): *A metalanguage for structural operational semantics*. In: *In Symposium on Trends in Functional Programming*, pp. 1–16. [cited at p. 190]

[98] Ruggero Lanotte & Simone Tini (2005): *Probabilistic Congruence for Semistochastic Generative Processes*. In Vladimiro Sassone, editor: *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, *Lecture Notes in Computer Science* 3441, Springer-Verlag, pp. 63–78, doi:10.1007/978-3-540-31982-5_4. [cited at p. 6]

[99] Huimin Lin (1995): *PAM: A Process Algebra Manipulator*. *Formal Methods in System Design* 7(3), pp. 243–259, doi:10.1007/BF01384078. [cited at p. 8, 168]

[100] Narciso Martí-Oliet & José Meseguer (2002): *Rewriting Logic as a Logical and Semantic Framework*. In Dov M. Gabbay & Franz Guenthner, editors: *Handbook of Philosophical Logic*, 9, Kluwer Academic Publishers, 2002, pp. 1–87, doi:10.1007/978-94-017-0464-9_1. [cited at p. 81]

[101] José Meseguer & Christiano Braga (2004): *Modular Rewriting Semantics of Programming Languages*. In Charles Rattray, Savi Maharaj & Carron Shankland, editors: *Proceedings of the 10th International Conference on Algebraic Methodology and Software Technology (AMAST'04)*, *Lecture Notes in Computer Science* 3116, Springer-Verlag, Berlin, Germany, 2004, pp. 364–378, doi:10.1007/978-3-540-27815-3_29. [cited at p. 81]

[102] Cornelis A. (Kees) Middelburg (2001): *Variable binding operators in transition system specifications*. *Journal of Logic and Algebraic Programming* 47(1), pp. 15–45, doi:10.1016/S1567-8326(00)00003-5. [cited at p. 191]

[103] Cornelis A. (Kees) Middelburg (2003): *An alternative formulation of operational conservativity with binding terms*. *Journal of Logic and Algebraic Programming (JLAP)* 55(1-2), pp. 1–19, doi:10.1016/S1567-8326(02)00039-5. [cited at p. 191]

[104] A.J.R.G. (Robin) Milner (1980): *A Calculus of Communicating Systems*. *Lecture Notes in Computer Science* 92, Springer-Verlag. [cited at p. 107, 137]

[105] A.J.R.G (Robin) Milner (1983): *Calculi for synchrony and asynchrony*. *Theoretical Computer Science (TCS)* 25, pp. 267–310, doi:10.1016/0304-3975(83)90114-7. [cited at p. 132, 143]

[106] A.J.R.G. (Robin) Milner (1989): *Communication and Concurrency*. Prentice Hall, Englewood Cliffs. [cited at p. 1, 3, 7, 8, 14, 21, 31, 41, 42, 108, 114, 131, 135, 150, 160, 168]

[107] Faron Moller (1990): *The Importance of the Left Merge Operator in Process Algebras*. In Mike Paterson, editor: *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16–20, 1990, Proceedings*, *Lecture Notes in Computer Science* 443, Springer, pp. 752–764, doi:10.1007/BFb0032072. [cited at p. 29]

[108] Peter D. Mosses (1999): *Foundations of Modular SOS*. In Miroslaw Kutylowski, Leszek Pacholski & Tomasz Wierzbicki, editors: *MFCS, Lecture*

Notes in Computer Science 1672, Springer, pp. 70–80, doi:10.1007/3-540-48340-3_7. [cited at p. 8]

[109] Peter D. Mosses (2004): *Exploiting Labels in Structural Operational Semantics*. *Fundam. Inform.* 60(1-4), pp. 17–31. [cited at p. 80]

[110] Peter D. Mosses (2004): *Modular structural operational semantics*. *J. Log. Algebr. Program.* 60-61, pp. 195–228, doi:10.1016/j.jlap.2004.03.008. [cited at p. 80, 98]

[111] Peter D. Mosses & Mark J. New (2009): *Implicit Propagation in Structural Operational Semantics*. *Electr. Notes Theor. Comput. Sci.* 229(4), pp. 49–66, doi:10.1016/j.entcs.2009.07.073. [cited at p. 94]

[112] Mohammad Reza Mousavi, Michel Reniers & Jan Friso Groote (2004): *Congruence for SOS with Data*. In: *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, IEEE Computer Society Press, Los Alamitos, CA, USA, 2004, pp. 302–313, doi:10.1109/LICS.2004.1319625. [cited at p. 80]

[113] Mohammad Reza Mousavi & Michel A. Reniers (2005): *Orthogonal Extensions in Structural Operational Semantics*. In: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, *Lecture Notes in Computer Science* 3580, Springer-Verlag, Berlin, Germany, pp. 1214–1225, doi:10.1007/11523468_98. [cited at p. 108]

[114] Mohammad Reza Mousavi & Michel A. Reniers (2006): *Prototyping SOS meta-theory in Maude*. *Electron. Notes Theor. Comput. Sci.* 156, pp. 135–150, doi:10.1016/j.entcs.2005.09.030. [cited at p. 161, 168, 169, 172, 173, 186]

[115] Mohammad Reza Mousavi, Michel A. Reniers & Jan Friso Groote (2004): *Congruence for SOS with Data*. In: *LICS*, pp. 303–312, doi:10.1109/LICS.2004.1319625. [cited at p. 7, 95]

[116] Mohammad Reza Mousavi, Michel A. Reniers & Jan Friso Groote (2005): *Notions of Bisimulation and Congruence Formats for SOS with Data*. *Information and Computation* 200(1), pp. 107–147, doi:10.1016/j.ic.2005.03.002. [cited at p. 7, 9, 13, 14, 15, 17, 80, 83, 92, 96, 182]

[117] Mohammad Reza Mousavi, Michel A. Reniers & Jan Friso Groote (2005): *A syntactic commutativity format for SOS*. *Inf. Process. Lett.* 93(5), pp. 217–223, doi:10.1016/j.ipl.2004.11.007. [cited at p. 79, 104, 109, 110, 111, 112, 125, 132, 133, 136, 138, 139, 145, 151, 179, 180]

[118] Mohammad Reza Mousavi, Michel A. Reniers & Jan Friso Groote (2007): *SOS Formats and Meta-Theory: 20 Years After*. *Theoretical Computer Science* 373(3), pp. 238–272, doi:10.1016/j.tcs.2006.12.019. [cited at p. 6, 7, 103, 105, 133]

[119] Scott Owens (2008): *A Sound Semantics for OCamllight*. In Sophia Drossopoulou, editor: *ESOP*, *Lecture Notes in Computer Science* 4960, Springer, pp. 1–15, doi:10.1007/978-3-540-78739-6_1. [cited at p. 80]

[120] David Michael Ritchie Park (1981): *Concurrency and Automata on Infinite Sequences*. In Peter Deussen, editor: *Theoretical Computer Science*, *Lecture Notes in Computer Science* 104, Springer, pp. 167–183, doi:10.1007/BFb0017309. [cited at p. 14, 21, 31, 83, 108, 131, 135, 174]

[121] Andrew M. Pitts (2003): *Nominal logic, a first order theory of names and binding*. *Information and Computation (I&C)* 186(2), pp. 165–193, doi:10.1016/S0890-5401(03)00138-X. [cited at p. 190]

[122] Gordon D. Plotkin (1981): *A structural approach to operational semantics*. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark. [cited at p. 167, 206]

[123] Gordon D. Plotkin (2004): *The origins of structural operational semantics*. *Journal of Logic and Algebraic Programming (JLAP)* 60, pp. 3–15, doi:10.1016/j.jlap.2004.03.009. [cited at p. 3, 80, 103]

[124] Gordon D. Plotkin (2004): *A structural approach to operational semantics*. *Journal of Logic and Algebraic Progamming (JLAP)* 60, pp. 17–139. This article first appeared as [122]. [cited at p. 3, 80, 103, 131, 159]

[125] Michel A. Reniers, Jan Friso Groote, Mark B. van der Zwaag & Jos van Wamel (2002): *Completeness of Timed μCRL*. *Fundamenta Informaticae* 50(3-4), pp. 361–402. [cited at p. 92]

[126] A. W. (Bill) Roscoe (1997): *The Theory and Practice of Concurrency*. Prentice Hall. [cited at p. 137]

[127] A. W. (Bill) Roscoe (2010): *Understanding Concurrent Systems*. Springer, doi:10.1007/978-1-84882-258-0. [cited at p. 79]

[128] Davide Sangiorgi (2011): *Introduction to Bisimulation and Coinduction*. Cambridge University Press. [cited at p. 138, 144]

[129] Davide Sangiorgi & David Walker (2001): *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, Cambridge. With a foreword by Robin Milner. [cited at p. 132, 137, 186]

[130] Traian-Florin Serbanuta, Grigore Rosu & José Meseguer (2009): *A rewriting logic approach to operational semantics*. *Information and Computation* 207(2), pp. 305–340, doi:10.1016/j.ic.2008.03.026. [cited at p. 169]

[131] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar & Rok Strnisa (2010): *Ott: Effective tool support for the working semanticist*. *J. Funct. Program.* 20(1), pp. 71–122, doi:10.1017/S0956796809990293. [cited at p. 190]

[132] Robert de Simone (1985): *Higher-Level Synchronizing Devices in MEIJE-SCCS*. *Theoretical Computer Science (TCS)* 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3. [cited at p. 7]

[133] Christopher Strachey (2000): *Fundamental Concepts in Programming Languages*. *Higher-Order and Symbolic Computation* 13, pp. 11–49, doi:10.1023/A:1010000313106. [cited at p. 86]

[134] Simone Tini (2004): *Rule Formats for Compositional Non-interference Properties*. *Journal of Logic and Algebraic Progamming (JLAP)* 60, pp. 353–400, doi:10.1016/j.jlap.2004.03.003. [cited at p. 6]

[135] Daniele Turi & Gordon D. Plotkin (1997): *Towards a Mathematical Operational Semantics*. In: *LICS*, IEEE Computer Society, pp. 280–291, doi:10.1109/LICS.1997.614955. [cited at p. 96]

[136] Irek Ulidowski (2000): *Finite axiom systems for testing preorder and De Simone process languages*. *Theoretical Computer Science (TCS)* 239(1), pp. 97–139, doi:10.1007/BFb0014317. [cited at p. 32, 103, 104, 125, 190]

[137] Alberto Verdejo & Narciso Martí-Oliet (2006): *Executable structural operational semantics in Maude*. *The Journal of Logic and Algebraic Programming* 67(1-2), pp. 226 – 293, doi:10.1016/j.jlap.2005.09.008. [cited at p. 169, 173]

[138] Chris Verhoef (1995): *A congruence theorem for structured operational semantics with predicates and negative premises*. *Nordic Journal of Computing* 2(2), pp. 274–302, doi:10.1007/BFb0015024. [cited at p. 7, 50]

[139] J. L. M. Vrancken (1997): *The Algebra of Communicating Processes With Empty Process*. *Theoretical Computer Science* 177(2), pp. 287–328, doi:10.1016/S0304-3975(96)00250-2. [cited at p. 14, 18]

[140] Axelle Ziegler, Dale Miller & Catuscia Palamidessi (2006): *A Congruence Format for Name-passing Calculi*. *Electr. Notes Theor. Comput. Sci.* 156(1), pp. 169–189, doi:10.1016/j.entcs.2005.09.032. [cited at p. 191]

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539