Contributions to the Meta-Theory of Structural Operational Semantics

Matteo Cimini
Doctor of Philosophy
November 2011
School of Computer Science
Reykjavík University

Ph.D. DISSERTATION

ISSN 1670-8539

Contributions to the Meta-Theory of Structural Operational Semantics

by

Matteo Cimini

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

November 2011

Thesis Committee:

Luca Aceto, Supervisor
Prof., Reykjavík University

Willem Jan Fokkink
Prof., VU University Amsterdam

Matthew Hennessy, Examiner
Prof., Trinity College Dublin

Anna Ingólfsdóttir, Co-Supervisor
Prof., Reykjavík University

MohammadReza Mousavi
Dr., Eindhoven University of Technology

The undersigned hereby certify that they recommend to the School of Computer Science at Reykjavík University for acceptance this thesis entitled **Contributions to the Meta-Theory of Structural Operational Semantics** submitted by Matteo Cimini in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

_____
Date

_____
Luca Aceto, Supervisor
Prof., Reykjavík University

_____
Willem Jan Fokkink
Prof., VU University Amsterdam

_____
Matthew Hennessy, Examiner
Prof., Trinity College Dublin

_____
Anna Ingólfsdóttir, Co-Supervisor
Prof., Reykjavík University

_____
MohammadReza Mousavi
Dr., Eindhoven University of Technology

Contributions to the Meta-Theory of Structural Operational Semantics

Matteo Cimini

November 2011

Abstract

Structural Operational Semantics (SOS) is one of the most natural ways for providing programming languages with a formal semantics. Results on the meta-theory of SOS typically (but not solely) say that if the inference rules used in writing the semantic specification of a language conform to some syntactic template then some semantic property is guaranteed to hold or some technique is applicable in order to gain some result. These syntactic templates are called *rule formats*.

This thesis presents four contributions on the meta-theory of SOS.

As a first contribution, (1) we offer a method for establishing the validity of equations (modulo bisimilarity). The method is developed under the vest of an equivalence relation that is suitable for mechanization, the *rule-matching bisimilarity*. Given a semantic specification defined in SOS and given the desired equation to check, the method runs a matching, bisimulation-like, procedure in order to determine the validity of the given equation. For the method to be applicable, the SOS specification must fit a well-known rule format called GSOS, which is fairly expressive. For instance most of the process algebras can be defined within GSOS. The method is general and, not surprisingly, might not terminate. We however show that relevant equations can be checked in finite time.

As another contribution, (2) we present rule formats ensuring that certain constants of a language act as zero elements. An example of zero element, though in the context of mathematics, is the number 0, that is a zero element for the multiplication operator $\times$, i.e., $x \times 0 = 0$. Based on the design of one of the formats, we provide also a rule format for unit elements. The same number 0 is for instance an example of unit element for the sum operator $+$, as the algebraic law $x + 0 = x$ is valid.

As a third contribution, (3) we offer rule formats guaranteeing the validity of the distributivity law. Examples of distributivity laws in the context of

mathematics are $(x + y) \times z = (x \times z) + (y \times z)$, i.e., the multiplication distributes over the sum, and $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$, i.e., the set intersection distributes over the union. The algebraic laws addressed by contributions (2) and (3) are considered modulo bisimilarity. In both contributions, the proposed rule formats are mostly mechanizable and some of them are also very simple to check. Nonetheless, the rule formats we offer are expressive enough to check the validity of classic zero and unit elements as well as well-known distributivity laws from the literature.

Thanks to contributions (2) and (3), now the meta-theory of SOS tackles all the basic algebraic laws, i.e., commutativity, idempotency, associativity, zero and unit elements and distributivity.

Finally, (4) we propose *Nominal SOS*, an SOS based framework with special syntax and primitives for the definition of languages with binders. Binders bind a name in a context in order to give it a certain meaning or to denote that a special treatment for it is required. The ordinary SOS framework lacks a dedicated account for binders. Binders, however, proliferate both in mathematics (one example is the universal quantification $\forall x.\Phi$) and in computer science (one example is the abstraction $\lambda x.M$ of the λ-calculus). We provide evidence that the framework is expressive enough to model interesting calculi. For instance, we formulated the λ-calculus and the π-calculus within the framework of Nominal SOS and we established the operational correctness of these formulations with regard to the original ones. We offer a suitable notion of bisimilarity that is aware of binding and we have embarked on a study of the relationship between this notion of bisimilarity and classic equivalences from the context of the λ- and π-calculus. We believe that the meta-theory of SOS is by now a mature field and times are ripe for a systematic study of the meta-theory that concerns also those phenomena that are specifically related to binders. In this respect, we believe that our framework might be a good candidate to carry out such a study.

Contributions to the Meta-Theory of Structural Operational Semantics

Matteo Cimini

Nóvember 2011

Útdráttur

Structural Operational Semantics (SOS) er eðlilegasta leiðin til að gefa forritunarmálum formlega merkingu. Niðurstöður er yfirkenningu fyrir SOS segja almennt (þó ekki alltaf) að ef málskipan (e. syntax) sem notuð er fyrir mekingarfræðilega (e. semantic) skilgreiningu máls fylgir tilteknu sniðmáti þá er ákveðnir merkingarfræðilegir eiginleikar tyggðir, eða að einhver tæknileg aðferð sé nothæf til að fá niðurstöður. Þessi málskipunar sniðmát (e. syntactic templates) eru kölluð regluform (e. *rule formats*).

Þessi lokariterð kynnir fjögur framlög til yfirkenningar fyrir SOS.

Sem fyrsta framlag, (1) kynnum við aðferð til meta lögmæti jafna (mótað við bisimilarity). Aðferðirnar eru þróaðar sem jafngildis vensl sem eru nothæf til sjálfvirknivæðingar sem *rule-matching bisimilarity*. Að gefnu merkingarfræðilegum skilgreiningum á SOS formi auk jöfnu sem á að prófa, gefur aðferðin matching, bisimulation-like, algrím til að ákvarða lögmæti jöfnunnar. Til að aðferðin sé nothæf verða SOS skilgreiningarnar að falla að vel þekktu formi GSOS reglna, sem er tiltölulega lýsandi. Sem dæmi er hægt að tákna flestar process algebrur með GSOS. Aðferðin er almenn en ekki er tryggt að útreikningum ljúki. Við sýnum þó að mikilvægar jöfnunur er hægt að prófa í endanlegum tíma.

Sem næsta framlag, (2) kynnum við regluform sem þar sem tilteknir fastar málsins virka sem núll stök. Dæmi um núll stak, í stærðfræðilegu samhengi er talan 0, en hún er núll stak fyrir margföldunarvirkjan $\times$, þ.e. $x \times 0 = 0$. òt frá þessu formi skilgreinum við regluform fyrir einingastak. Sama talan, það er 0, er dæmi um einingarstak fyrir summu virkjan $+$, þar sem algebru reglan $x + 0 = x$ heldur ávalt.

Sem þriðja framlag, (3) kynnum við regluform sem tryggir dreifni-lögmálið. Dæmi um dreifni-lögmálið í stærðfræðilegu samhengi er $(x + y) \times z = (x \times z) + (y \times z)$, það er margföldun dreifist yfir summu, og $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$, það er sniðmengi dreifist yfir sammengi. Algebru lögmálin

sem kynnt eru í (2) og (3) eru skoðuð mótuð við bisimilarity. Í báðum framlögunum eru regluformin sem kynnt eru að mestu leyti hægt sjálfvirknivæða og sum þeirra er einfalt að sannreyna. Þrátt fyrir það, eru reglu formin nógu lýsandi til að kanna lögmæti klassískra núll- og einingastaka, og vel þekktra dreifni-lögmála úr fræðunum.

Með framlagi (2) og (3) þá er yfirkenning SOS fær um að leysa öll grunn algebru lögmálin, það er, víxlun, sjálfvöldun, tengireglu, núll og einingarstaka auk dreifireglu.

Að lokum, (4) kynnum við *Nominal SOS*, sem er rammi byggður á SOS með tilteknum syntax og primatives fyrir skilgreiningar á máli með binders. Binders binda nöfn í samhengi til að gefa merkingu eða tákna þó að þörf sé á sérstakri meðferð. Heffbundin SOS rammi skortir skilgreiningar sem lýsa binders. Binders er að finna víða í stærðfræði (dæmi um binder er allsherjarvirkinn $\forall x.\Phi$) og í tölvunarfræði (sem gæmi má nefna sértekninguna $\lambda x.M$ í λ-calculus). Við sýnum að raminn er nægjanlega lýsandi til að módela áhugaverð mál. Sem dæmi þá formuðum við λ-calculus og π-calculus innan ramma Nominal SOS og við sýndum fram á lögmæti formanna með tilliti til upprunalegu formanna. Við kynntum viðeigandi táknun á bisimilarity og klassískra jafngilda í samhengi λ- og π-calculus. Við teljum að yfirkenning SOS sé nú orðið þroskað svið og tími sé tilkominn til að rannsaka yfirkenningu í samhengi hluta sem tengjast binders. Í þessu samhengi þá teljum við að ramminn sem kynntur er sé henntugt kerfi til að framkvæma slíkar rannsóknir.

Preface

> *Twenty years from now you will be more disappointed by the things you didn't do than by the ones you did do. So throw off the bowlines, sail away from the safe harbor. Catch the trade winds in your sails. Explore. Dream. Discover.*
> Mark Twain.

> *Like all great travelers, I have seen more than I remember, and remember more than I have seen.*
> Benjamin Disraeli.

> *This was really the way my whole road experience began, and the things that were to come are too fantastic not to tell.*
> Jack Kerouac, from the book *On The Road*.

> *So I asked my gut, and it said that the compass was pointing North.*
> From Matteo Cimini's Ph.D. Thesis Preface.

After my Master's Degree I was faced with a looming question: What will I do? I knew I wanted to do research on some theoretical aspect of computer science, and perhaps get a Ph.D. along the way. I also knew myself well enough to realize that I was on the hunt for something different, surely an experience abroad.

I have to admit that, at the time, moving to Iceland was beyond my wildest expectations. Of all the twists and turns of my life, winding up in Iceland ranks among the most special. I believe that mine was, at very best, a non conventional decision. This preface is about this choice and briefly of what I ventured into by living this decision for the past three years.

Searching for a job, I eventually came across an announcement by Luca. He and Anna opened a position as a Ph.D. student in one of their projects. It was at the Reykjavik University. The subject matched perfectly my interests, I did not know Luca or Anna personally but I knew their work and I heard only extremely good

words about them. I however recall that I did not know much about Iceland. Reminiscing now through the foggy memories I have of that time, I would say that I applied for the position with almost no hesitation and with a light mind, delegating to a future Matteo the actual dealing with a possible acceptance.

When Anna and Luca informed me that the position was mine if I wanted, a million thoughts raced through my mind. To add further complexity to my doubts, another great job offer came along in the meanwhile. But I will not get into the latter matter, as though the main deal of that moment still was whether I was willing to throw myself into the Icelandic adventure. And we will get quite a long way talking of this.

Committing to a 3-year long Ph.D. program is not a joke. Furthermore, committing to living long-term in a country like Iceland is something you have to mull over with care.

As per usual, when faced with a high-stakes decision, you want to think things through. It was no time for snap decisions.

I honestly had little knowledge about Iceland that time. It was way back in early summer 2008 and Iceland did not appear much in Italian news papers by then, I guess. Iceland had always been something of a mysterious and unknown land for me, and I can fairly say that people back home would second this feeling, too. This is witnessed by the countless bizarre questions that I am always asked about this country.

The big Icelandic economic default was on its way, and would take place a couple of months later. In a way, it is only fortunate that the crisis came up later than my decision. In all likelihood, it would have played a negative role in my decision process and I might have ended up being too reluctant to go, missing out on one of the most valuable experiences of my life.

Knowing so little about Iceland, I began a thorough search and I also called up some people who happened to have lived there for a short while. I was craving for information. I heard incredible things and many misconceptions just fell apart. I had a feeling that much was there, yet to be discovered and just willing to be seen, whatever it was.

I had never lived abroad and Iceland was quite a long shot as a first try. This country appeared to be light years different from Italy, it was a big jump for me, a jump to an unfamiliar and remote world. I envisioned complexities of higher

levels. If I were to take a map, Iceland would seem just a spot in the middle of nowhere.

Inevitably, my mind would peg this place as mysterious and foreign as outer space. And for as much as I tried to picture myself in Iceland, still I could not develop the slightest hunch as to what my life could look like once there. If I looked at my crystal ball I could see only foggy and hazy images. No, the "picture yourself in Iceland" game was not working.

So there I was, with a huge life-decision to make, and despite all the search in the world no clear idea was shining through to me. For a time, I was racking my brain over this matter and, I have to say, not a few people would find it bizarre that I was even considering the idea of moving to Iceland.

But sometimes you take a leap into something that is neither risk free nor popular. Somehow, a wave of instinctive attraction towards this little country was pulsing its way into my head. I was divided but this thought was inescapable and I wanted to see what life had lined up for me in such a remote, yet fascinating, land. So I asked my gut, and it said that the compass was pointing North. I took the leap and I said to Luca and Anna to wait for my arrival. A few months from there I was in Iceland.

Next thing I know, I was in Iceland indeed. It was way back in January 2009, and my journey to the unfamiliar just began, right there.

"Different" became the byword of my Icelandic adventure since the very first day. Iceland really is different from anywhere else I have been. In a space of three years, I could enjoy landscapes of unmatched beauty and natural phenomena that would shake me to speechlessness.

The reader might have noticed that the quotations that open this preface are about the very concept of travelling. In some sense, Iceland has been a long voyage for me.

But I have to say, my real journey had little to do with touristic sightseeings, and a whole lot to do with the people I met along the way and the experiences I had with them. Living the culture and the dynamics of this country, too, has been a core part of this journey.

Admittedly, Iceland might not be easy to live in sometimes. This is especially true for someone coming from a completely different country, as Italy is indeed. There are many things I had to learn to roll with. But as years rolled around, I

grew fond of this little country in its entirety and, magically enough, that one non conventional decision led me to one of the greatest experiences of my life.

Had I known that it would have all worked out the way it has, I would not have had a shred of doubt in my mind.

Acknowledgements

Even though the cover of this book displays only one name, many people contributed to the existence of this thesis, whether directly or not. This is my chance to thank them properly.

The first two persons I am itching to thank are Anna Ingólfsdóttir and Luca Aceto. Finally I can do it black and white. I set my first foot in this jungle called "academic world" with them and I did not know exactly what to expect. We walked this long road towards my Ph.D. together, they coached me through every step of the way and taught me a multitude of things. It has been an honor to be their first Ph.D. student. With their knowledge and experience they have been an extraordinary guide for me. Moreover, their passion for their work has always been contagious and motivational and they always provided me with warm encouragement and support in countless occasions and in countless ways. I am indebted to them more than they know and I am only happy to finally write a capitalized THANK YOU for them.

I defended my thesis on the 18th of November, in one of the longest days of my life. Besides my supervisors, the members of my Thesis Committee are Matthew Hennessy, Wan Fokkink and MohammadReza Mousavi. I feel very honored for having had such an excellent committee. I thank them for their thoughtful comments and for finding me worth a Ph.D. title, turning a day that on calendars appears as a November Friday like any other into such a special day for me. They have all my gratitude.

I am grateful to Dale Miller and Andrew Pitts. The work presented on Chapter 5 is strongly related to work they have carried out and their comments improved significantly the related work section of that chapter.

I thank Reykjavik University in its entirety. The university provided me with an excellent environment where I could work in freedom. The first adjectives that spring to my mind about my work environment are: beautiful, peaceful

and stimulating. During these past three years I really could work under perfect conditions. Reykjavik University is one of those universities you just feel proud of being part of. The new premise of the university is a beautiful majestic building. Various research projects are going on in there and you deal with interesting people and with their visions. The university always has been an oasis of ideas and nowhere I found my work inspired more than in there. Living through the dynamics and the changes that the university underwent over these years has been interesting and stimulating, and I have been lucky to have had wonderful colleagues I could share all of this with.

I met a number of people at Reykjavik University that I would like to acknowledge. During my Ph.D. time I inevitably had a number of bureaucratic or technical matters to handle. Although all the university offices have been helpful to me, I wish to thank especially two girls who never failed to promptly assist me: Kristine Helen Falgren and Sigrún María Ammendrup. Ari Kristinn Jónsson, currently president of Reykjavik University, was the Dean of the Computer Science School when I first started my Ph.D. path. He never failed in being helpful and letting me feel welcome since the very first day. I am also thankful to Marjan Sirjani for involving me in her research activities and for being nice and playful company any time. Also, the current Dean of the Computer Science School, Björn Þór Jónsson, has been precious to me a number of times.

At one point within the space of these three years I spent many months in the Netherlands. The purpose of such a stay was a collaboration with MohammadReza Mousavi and Michel Reniers in Eindhoven. I thank them for this opportunity. Living in the Netherlands has been a valuable experience. Working with them at Eindhoven University of Technology has been a fruitful and also fun time. I also thank Tineke van den Bosch, the secretary of the Department of Mathematics and Computer Science, for being precious with regard to logistic matters. I remember and greatly cherish the interactions I had with Bas Lutik, Erik De Vink, Jan Friso Groote, Jos Baeten, Matthias Raffelsieper, Paul Van Tilburg and Tim Willemse.

My time in the Neatherlands would not have been as joyful as it had been if it were not for a group of Italians I met there. I spent a nice time with Antonio Corradi, Alberto Nucciarelli, Daniela Scalise, Daniel Trivellato and Elisa Costante. I am also greatly thankful that during my Dutch stay I spent time with Paul Van Tilburg and his friends, and also some with Alexandra Silva.

My indebtment to people goes further back than my Ph.D. time. As I am writing these acknowledgments I find myself in Bologna for a short visit. Bologna is the city where I had lived for many years and where I received both my Bachelor's and Master's degrees. At the University of Bologna. I am grateful to those years and to those professors who molded me the most on my way to my graduations. In particular, I send special thanks to Davide Sangiorgi and Claudio Sacerdoti Coen.

Also, I would like to mention that it was a nice surprise to find out that the School of Computer Science at Reykjavik University has an agreement for double graduations with the Department of Computer Science at the University of Camerino. I did not know this until I came to Iceland and it was a surprise because this university is really close to my home town in Italy. Because of this agreement, over the years I have met a number of students coming from the University of Camerino and mostly from around my place in Italy, the beautiful region Marche. It also has been nice to meet every now and then people from that department, in particular Emanuela Merelli, Luca Tesei and Nicola Paoletti.

I have met many other people who have been a positive note during these years. I find it difficult to place them in categories, and also I am not sure I want to. Some of them are great friends I already had back in Italy, some of them are friends I made along the way, some are even great companions of adventures and experiences while others simply affected positively a short part of this long journey and that for some reasons I would like to acknowledge. Whether in the little or big way, they all contributed to making this experience one such that trumps any other I had so far in my life and they deserve resonating thanks. My gratitude goes to Alessio Ciricugno, Alisa Widmer, Andrea Ellen Jones, Angelo Cafaro, Anu Palomäki, Ari Þór Arnbjörnsson, Arnar Birgisson, Bruno Fanini, Candy Caldwell, Carlos Gregorio-Rodríguez, Carlos Hernandez Corbato, Carmine Oliva, Chris Severs, Claire Johnstone, Claudio Pedica, Daniella Gullans, David de Frutos-Escrig, Emanuele Travanti, Enrico Rossomando, Eugen-Ioan Goriac, Fabio Marziali, Federico Buti, Francesco Stablum, Georgiana Caltais, Giuseppe Esposito, Giuseppe Servici, Hannes Högni Vilhjálmsson, Ingibjörg Vagnsdóttir, Jacopo Penazzi, Joshua Sack, Luca Mandrioli, Manuela Berardi, Marco Ciaschini, Maria Guidi, Mario Bravetti, Mark Dukes, Martina Patone, Natalia Jerzak, Niccolò Rossetti, Pradipta Mitra, Raffaele Gaito, Riccardo Pancotti, Robert Parviainen, Rolanda Simenaite, Sabrina Stiegler, Shishir Patel, Stefanía Helga Stefánsdóttir, Stephan Schiffel, Steve Widmer, Tom Matthews, Ute Schif-

fel, Verena Steinbauer, Victor Carazo Robles, Viðar Hrafnkelsson and Vincenzo Negrone.

English words, and words at all for that matter, fail me to express my gratitude to my parents Giampalma and Sergio and to my brother Gianluca. **Il più grande grazie va davvero a voi.**

Contents

Chapter 1

Introduction

"Begin at the beginning,", the King said, very gravely, "and go on till you come to the end: then stop.".
Lewis Carroll, from the book *Alice's Adventures in Wonderland*.

1.1 Semantics

The word *semantics* generally denotes the science of the meaning communicated through language. The most established areas that are related to semantics are linguistics and philosophy, where it concerns the study of the meaning of linguistic expressions in a natural language.

As an example, the reader may consider the sentence "It is raining." This sentence is nothing but a piece of syntax formed by a sequence of well-formed words from a vocabulary. This syntactic object has no meaning *per se* and is of the same nature as any other mere syntactic expression the reader may think of, for instance the mathematical equation

$$\sum_{i=1}^{i=n} i = \frac{n * (n+1)}{2}.$$

Semantics gives a meaning to these sequences of symbols. Indeed, the meaning is the actual item we want to express, and language is only one vehicle by which we (try to) express it. In particular, with the above sentence "It is raining." our aim is to state a subsisting fact upon the actual world we live in and we say that this sentence is true if, looking at the actual world, it really is raining.

Natural language is however a very elusive and somehow deceptive monster to tame, and although this example might make it seem as though its semantics analysis is simple, the reader surely would have no difficulty in realizing that the way we communicate with one another is conversely full of varieties and shades. The study of the semantics of natural languages is indeed a difficult research area.

In computer science, languages proliferate too, and they are artificial tools whose purpose is describing computations that a computer can perform (programming languages) or modelling some scenario or ontology of interest (specification languages). Assuming that the reader is familiar with common imperative programming, let us consider the following piece of pseudo-code.

```
while (i <= n)
    x=x+i;
    i=i+1;
```

Again, the reader may notice that the piece of program above is merely a syntactic object, I might have used the expression "as long as" in place of "while" if I wished, or I might have used the symbol $*$ in order to denote arithmetic addition, in place of the standard symbol +. What matters at the very end is the meaning we associate with all of these syntactic entities. In this case we would want the meaning of the above piece of program to express that

> if the variable i is less than or equal to the variable n, sum the value of the variables x and i and store the result as value of the variable x, increment the value of the variable i by one, and repeat this procedure as long as i is less than or equal to n.

Unlike natural languages, programming and specification languages are given a well-established syntax, provided by means of a formal grammar. Moreover, while the semantics of a natural language is perforce inferred *a posteriori* via its use, the semantics of programming languages must be specified *beforehand* and is a vital part of a language specification in order to have the possibility to implement the language on a computer and also to obtain a common understanding of it among its users.

In the early days of computer science, the semantics of programming and specification languages was often left to informal natural language descriptions, which may be ambiguous and leave open questions to their implementors and users.

In contrast to these informal semantic descriptions, programming and specification languages can instead be given a *formal semantics*, which provides their expressions with a precise meaning. Providing a formal semantics is moreover useful because it associates some kind of mathematical structure to programs, opening the possibility of performing mathematical analysis of their behaviour. As software is omnipresent in our lives, this turns out to be very important nowadays. Indeed, being software embedded in airplanes, medical devices and in many other delicate contexts, program errors may cause loss of lives, money and even may have environmental impact. Having techniques to reason formally about the correctness of programs is therefore of increasing importance, and this is possible only if languages are provided with a formal semantics.

It is thus not surprising that the field of semantics has received considerable attention from the computer science community since the very beginning. Over the years, many approaches and techniques have been developed in order to give semantics to computer languages, and the study of the semantics represents without doubt an important and exciting area of research in computer science, which is still evolving.

As the reader may expect, despite the great effort devoted to the study of techniques for providing formal semantics to programming languages, no silver bullet has been found and a number of theories of semantics have been developed over the years, each of which coming with its own peculiarities and pragmatics.

One of the most successful theories of semantics is the so called *operational semantics*. This is the general subject of my thesis and the next section is devoted to a brief presentation of the ideas behind this method. There are of course other methods which may be used to give formal semantics to programming languages; for overviews of this field, we refer the reader to some excellent general textbooks such as [113], [152], [70], [133], [73] and [80], just to name a few.

1.2 Structural Operational Semantics

Operational Semantics In operational semantics the execution of a program is described by the series of steps it can take. More precisely, the computation is described by a transition system whose states describe the configurations a program might be in during its execution and whose transitions are the computation steps possible from one state to another. Transitions may be labelled. The label

of a transition yields information on the computation step/action that caused the transition. An example of transition is

$$(x := 3; P, \sigma) \xrightarrow{\tau} (P, \sigma[x \mapsto 3]),$$

which must be read as *from the state* $(x := 3; P, \sigma)$ *a computational step labelled* τ *can be performed, ending up in the state* $(P, \sigma[x \mapsto 3])$.

Intuitively, the state of the system is a pair formed by a program and a store σ, which keeps track of the associations variable-value. In this case $x := 3; P$ is the program, which is an assignment of the value 3 to the variable x followed by the evaluation of the program P. The operator ; is the standard sequential composition of programs. The label τ for transitions conventionally denotes an internal progress of the system in the world of process algebra. We employed this label as the transition represents the execution of a statement which has no interaction with the external environment. As we would expect, with the above computation step we end up in a state where the program to evaluate next is P and it must be evaluated in the environment σ modified so that the variable x is mapped to the value 3.

Structural Operational Semantics During my thesis work I contributed to a particular approach to giving operational semantics to programming and specification languages, the so-called *Structural Operational Semantics*, SOS from now onwards, and in particular, I contributed to its meta-theory.

In 1981, Gordon Plotkin introduced SOS in his seminal lecture notes [119] to give a logical means to defining the operational semantics of programming languages. In this approach, the operational semantics is specified by a series of inference rules whose aim is to prove transitions. Rules are of the form

$$\frac{\textit{premises}}{\textit{conclusion}}$$

Roughly speaking, a rule like the one above means that if the premises are satisfied then the conclusion, which must be a transition, can be proved. Typically, the inference rules are used to describe the behaviour of the operators of the language at hand, and this is typically done in terms of the behaviour of its arguments.

For the sake of concreteness, the reader may want to consider the following example containing the formalization of a simple concurrent language.

Example 1.2.1 *We assume a set Act, the actions that processes can perform. We consider a language with parallel composition, denoted by ||, non-deterministic choice, denoted by +, action prefixing a._ for each a ∈ Act and the inaction constant* **0**. *The informal behaviour of the operators is as follows.*

- *The operator a (one for each a ∈ Act) is unary, and the process a.P performs the action a and then executes the process P.*
- *The operator +, called the choice operator, is binary, and the process $P_1 + P_2$ behaves either like P_1 or P_2; the choice is made upon the performance of the first action of either P_1 or P_2.*
- *The binary operator ||, called the parallel operator, is binary, and the behaviour of the process $P_1 \,||\, P_2$ is the interleaving of those of P_1 and P_2.*
- *The operator* **0** *has arity 0, i.e. it is a constant of the language. It stands for the process that does not make any computation step.*

The generated grammar of such a language is thus:

$$P = \quad \mathbf{0} \mid a.P \mid P_1 \,||\, P_2 \mid P_1 + P_2,$$

where a ∈ Act.

The informal behaviour of the operators is formalized by means of the following inference rules.

$$\frac{}{a.x \xrightarrow{a} x}$$

$$d_1 = \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad d_2 = \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

$$\frac{x \xrightarrow{a} x'}{x \,||\, y \xrightarrow{a} x' \,||\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,||\, y \xrightarrow{a} x \,||\, y'}$$

The reader should notice that the actions are employed as labels for transitions and that the constant **0** *affords no transitions. Indeed, no rule defines its behaviour.*

In order to prove transitions for a process, we can use the inference rules above. Consider for instance the process $a.\mathbf{0} + b.\mathbf{0}$. We can pattern match the process

$a.\mathbf{0} + b.\mathbf{0}$ with the term $x + y$ of the conclusion $x + y \xrightarrow{a} x'$ of the rule d_1 and with the term $x + y$ of the conclusion $x + y \xrightarrow{a} y'$ of the rule d_2. The reader can easily see that these are the only options when proving transitions for $a.\mathbf{0} + b.\mathbf{0}$. Let us consider using the rule d_1. The pattern matching is successful, when x is mapped to $a.\mathbf{0}$ and y is mapped to $b.\mathbf{0}$. In order to apply the rule d_1 we must verify that its premise $x \xrightarrow{a} x'$ is satisfied. This amounts to verifying whether the process $a.\mathbf{0}$ is able to perform a transition labelled a. Repeating the process, we can see that the axiom $a.x \xrightarrow{a} x$ can be used to prove the transition $a.\mathbf{0} \xrightarrow{a} \mathbf{0}$, more precisely when the variable x is mapped to $\mathbf{0}$. Thus, the premise of d_1 is satisfied when we also map x' to $\mathbf{0}$, and the rule can be used in order to prove the transition $a.\mathbf{0} + b.\mathbf{0} \xrightarrow{a} \mathbf{0}$. Applying the same reasoning, considering the rule d_2 this time, we can prove the transition $a.\mathbf{0} + b.\mathbf{0} \xrightarrow{b} \mathbf{0}$. These are the two transitions that we would expect from the process term $a.\mathbf{0} + b.\mathbf{0}$.

The reader may see that SOS is a restriction on the more general operational semantics. In SOS we commit to calculating the steps of programs by means of an inference system whose rules are based on premises and conclusions.

This restriction should not be seen as an actual limitation by the reader. Rather, it is a useful way to structure the use of the operational semantics. Indeed, many benefits come from restricting ourselves to SOS, and these benefits are coming while retaining great expressiveness in modelling interesting calculi.

First of all, the premises/conclusion language provided by SOS is a natural, clean and flexible approach to model the semantics of languages. The reader may consider again the examples of the choice and parallel operators, which clearly show how naturally the semantics of these operators is formulated within SOS. These operators are formalized not only easily, but, in some sense, also in the way they are supposed to be formalized. This is possible thanks to the inductive nature of the rules. Gordon Plotkin, commenting on the SOS rules in their early stage of development, says in [120] that:

> I saw the [SOS] rules as directly formalizing the natural English description...

Also Milner, in his first book on CCS, [96], where the structural operational semantics for that calculus was first published, remarked that the original definition of Algol68, though strongly verbal, is in essence a set of reduction rules.

Not only with SOS can the user formulate the semantics easily, but, equally importantly, SOS is easy enough to allow users not to have difficulty in reading and understanding the semantics of languages written by others.

From a technical point of view, one of the main remarkable benefits of this syntax driven approach is that proof techniques based on induction turn out to be a sufficient tool to prove many of the relevant properties about the evolution of programs.

Even though SOS is a restricted discipline within the general framework of operational semantics, still the literature shows it to be an expressive way to define the semantics for many languages, see [99, 121, 97, 1, 48, 55] for a few examples.

1.3 Meta-theory of SOS

Due to its popularity, SOS became quickly itself a subject of investigation in the research community. This led to the birth of the meta-theory of SOS.

According to Encyclopædia Britannica Online[1], a meta-theory is

> a theory the subject matter of which is another theory.

Given a mathematical theory, we can use it to prove facts *within* the theory, proving the so-called *theorems*. Often, it is fruitful to consider the theory itself as the subject of investigation. The main point is to take a more abstract viewpoint and look from a higher vantage point at the considered theory in order to draw general statements about the way it works and find regularities and laws. These laws are called *meta-theorems* and they state general facts about the theory considered.

While the concept of meta-theory has always been addressed unconsciously by mathematicians, we can fairly attribute to David Hilbert a first clear separation of the distinction between theory and meta-theory. He was the first mathematician to raise, in his famous list of 20 problems in 1900, meta-reasoning questions about mathematics.

Namely, he proposed to the community the use of mathematical tools in order to investigate the formal systems employed in doing mathematics itself. Since Hilbert, the mathematical community has been engaged in a fruitful and exciting path to the discovery of very basic truths about the expressiveness and limits of

[1] http://www.britannica.com/EBchecked/topic/378037/metatheory

formal systems. These meta-reasoning over formal systems led to meta-theorems that shook the foundations of mathematics and led to the development of computer science [50].

For the sake of concreteness, perhaps the reader may want to consider one of the most famous problems posed by Hilbert to the mathematical community, which concerns Peano arithmetic, formulated by Giuseppe Peano in [116][2]. This is a striking example that clearly explains the distinction between theory and meta-theory. Peano arithmetic forms the ordinary arithmetic adopted so far in mathematics and consists of a set of axioms which is *closed under* some notion of inference rule. In particular, the axioms of the theory state basic facts, i.e. the ordinary definitions of natural numbers, sum and multiplication, that the number 0 cannot be successor to any number, that if two numbers have the same successor then they are the same number, and the induction principle. These axioms are used along with the standard inference rules of logic and equational reasoning.

The reader can see that by playing the game of applying the inference rules starting from the axioms, and so in a sense by playing the game from *within* the system, we can manipulate sentences syntactically and see which other sentences we can end up with. These are the theorems that we can prove in Peano arithmetic. For instance, we can prove the commutativity and associativity of the sum operation and, by a more tortuous path of applications of the rules we can also prove the Fundamental Theorem of Arithmetic, and so on, along this line, we can derive other theorems of arithmetic.

One of David Hilbert's question was whether Peano arithmetic could prove its own consistency, namely that it is not the case that applying the inference rules, the system is able to prove a theorem and, by some other proof path, the negation of the same theorem. This is an example of a meta-theoretic question. Now, the question is indeed *about* Peano Arithmetic and the way it works. In order to answer this question, Peano Arithmetic itself is to be investigated with some mathematical tool in order to understand its peculiarities. As we know, Gödel answered this question negatively in his second incompleteness theorem, [79].

Another example of distinction between theory and meta-theory comes from geometry, in particular from Euclidean geometry. In its original formulation by Euclid, given more than 2000 years ago, this theory consists of 5 postulates

[2] The reader can find an english translation of [116] in [148].

and a number of *common notions*. They are basically what we call axioms and inference rules, and, remarkably, it is an early example of formal system capable to rigorously, almost syntactically, derive the theorems of geometry. Hilbert noticed that, as rigorous as Euclid's formulation may appear, it is still not syntactic enough to let one play a mechanical, say merely syntactical, transformation of sentences in order to derive theorems. One of the many contributions by Hilbert to mathematics is a formalization of Euclidean geometry as a formal system in this strict sense, in [75].[3]

Again, applying the inference rules starting from the axioms of geometry, so by playing the game from within, we are able to prove the theorems of geometry, such as the famous Pythagoras' theorem among others. From the outside, anyway, many questions are interesting *about* geometry. As a striking example, for instance, mathematicians quickly realized that one of the postulates of the theory, namely the 5-th postulate, is of a different nature. This postulate, called the parallel postulate, states (in an equivalent formulation) that two parallel lines would never intersect each other in any point. As simple and self-evident as it may appear, this axiom is far more complicated than the other axioms of the theory. A natural question was thus whether this axiom would be somehow a fact already implied by the other axioms or not.

One of the most important meta-theorems in geometry is that the parallel postulate is actually an independent axiom, i.e. it cannot be proved by the other axioms and moreover, keeping the other axioms, adding the parallel postulate or adding the negation of the parallel postulate (there are several ways to negate the 5-th postulate) leads to alternative consistent theories of geometry. These new geometries are the so-called Non-euclidean geometries.

The book [79] contains an excellent exposition on the distinction between theory and meta-theory and it also surveys with elegant and easy to understand language the example of Peano arithmetic we discussed previously.

The development of the Meta-theory of SOS Getting back to the subject of my thesis, we recall that SOS is a theory of semantics. Given a semantics it can be used to prove (*within* the theory) the transitions that programs can perform. Based on the transition-system semantics, just as in the previously mentioned examples, SOS can also be looked at from the outside and become the subject of analysis,

[3] Other well-known axiomatizations of Euclidean geometry are those of Alfred Tarski, [134], and of George Birkhoff, [41].

which leads to the so-called Meta-theory of SOS. Investigating the meta-theory of SOS has proven to be particularly fruitful and the reader may find in [18] and [109] two broad, although a little dated, overviews of the subject.

The first step towards meta-reasoning about a theory is having a formalization of it, i.e., a formal account which may be used as the subject of some kind of mathematical investigation. In [68], Jan Friso Groote and Frits Willem Vaandrager formalized SOS by means of the notion of *Transition System Specification*, TSS from now onwards. TSSs turn out to be a powerful tool to be used and a flexible formalism to analyze. Technical definitions are left to the chapters to follow, but intuitively, a TSS describes a semantics for a language by means of a triple that contains

- a *signature*, i.e. the set of operators of the language together with their arity,
- a set of labels of transitions, and
- a set of deduction rules.

For instance, in the TSS formalizing the language in Example 1.2.1, the signature contains the constant $\mathbf{0}$ (with arity 0), the unary operator a, and the two binary operators + and $\|$, the set Act is the set of labels, and the reader can see the set of rules defined within Example 1.2.1.

The most important meta-theorems of SOS concern syntactic restrictions of TSSs that are able to ensure some semantic properties of the induced semantics or that are suitable as the target of proof techniques for gaining relevant results. These restrictions are called *Rule Formats*. A list of the main types of meta-results in SOS is presented below.

- Congruence of behavioural equivalences. The literature offers rule formats guaranteeing that some behavioural equivalence or preorder between terms is a congruence. An equivalence relation R is a congruence with respect to a function symbol f of arity n whenever, given two sequences of n closed terms $\vec{P}$ and $\vec{Q}$, if $\vec{P}\ R\ \vec{Q}$ then $f(\vec{P})\ R\ f(\vec{Q})$. The relation R is a congruence when it is a congruence with respect to f, for any function symbol f in the signature. Congruence is a very important property for an equivalence relation, because it means that we can safely substitute equals for equals no matter which context they appear in. Congruence formats w.r.t. bisimilarity are GSOS [44], Tyft/Tyxt [68], NTyft/NTyxt [66], Path [31], Panth [150], and the promoted Tyft [40], just to mention some. Bisimilarity is not the only

equivalence relation taken into account, and also several preorders have been addressed so far, the reader is invited to consult [109] for a survey on these results.

- Algebraic properties. The literature offers rule formats addressing most of the common algebraic properties of language constructs. For example, in [110] the authors provide a rule format guaranteeing the commutativity of certain operators (modulo bisimilarity). In [4] a rule format for the idempotence of operators is provided. Associativity of operators is addressed in [49] and the existence of right and left unit elements for binary operators in [22].

- Global properties of the computations of programs in a language. As examples, in [60] the authors provide a rule format guaranteeing the bounded non-determinism of the induced semantics and in [4] the authors offer a rule format ensuring the determinism of certain transition relations.

- Connections with denotational models of languages. The literature offers methods capable to automatically generate denotational models starting from an SOS specification. In order to be useful, such models are required to be fully abstract, in the sense of [94, 118, 139], with respect to a behavioural equivalence or preorder of interest. The mentioned methods are by and large possible only when TSSs are restricted to fit particular rule formats. Seminal contributions along this line are due to Bard Bloom [42] and also Jan Rutten and Daniele Turi [126, 127]. Successive contributions are [19], [128], to mention some. As an example in [19] the authors provide a method to automatically generate CPO-based models for a particular subclass of GSOS languages [44]. The domain of such models is Samson Abramsky's domain of synchronization trees [3] and models are proved to be fully abstract with respect to the bisimulation preorder.

- Conservative extension. Given a programming language, it is often interesting to add new operators to it. In the SOS world this means extending the TSS for the original language with new function symbols together with the rules describing their behaviour and/or with new rules for existing operations. A conservative extension of a TSS adds new operators and/or rules without changing the behaviour of the terms of the old language. Checking whether an extension is conservative is particularly desirable in order to benefit from the results concerning the old language. The reader is invited to consult [17] and [59] for excellent overviews of the subject.

- Proof systems for Hennessy-Milner logic. Hennessy-Milner logic (HML) [74] is a modal logic whose formulae are intended to specify computational properties of the states in labeled transition systems. Repeating its syntax and semantics is out of the scope of this brief description. Still, it is sufficient for the reader to think that typical properties specified by HML are
  - Is the current state capable of performing an a-action?
  - Is it possible, from the current state, to perform a step and reach a state that is capable of performing an a-action?
  - Is every possible state I can reach by performing a step from the current state capable of performing an a-action?

  Since HML formulae state properties about the behaviour of programs, proving whether a formula is valid is of particular interest. It is thus not surprising that checking the validity of HML formulae has been addressed extensively in literature. The subject is independent from the meta-theory of SOS; nevertheless, Alex Simpson linked the two fields in [138], offering an algorithm for generating a sound and complete proof systems for such a logic in the context of languages whose semantics is specified in the GSOS format.

- Equational theory axiomatizations. An equational theory aims to characterize a congruence relation by means of a set of axioms using the inference rules of equational reasoning. An equational theory is sound with respect to an equivalence relation $\approx$ whenever all the terms equated by the theory are indeed equated by $\approx$, i.e. the equational theory cannot prove false/invalid equations. Conversely, an equational theory is complete whenever all the equivalences between terms that hold modulo $\approx$ can be proved equal by the theory. Often ground-completeness is also addressed, which is completeness restricted to closed terms. Given a fragment of a process calculus, it is common practice to provide an axiomatization of some behavioural equivalence of interest over it. The reader may think about the example of such a practice provided by Hennessy and Milner in [74] for the language CCS modulo bisimilarity, see also [96]. Since then, there have been a plethora of results pertaining to the (non-)existence of finite equational axiomatizations for behavioural equivalences. The positive results on the existence of "nicely specified" ground complete axiomatizations often rely on normal forms for terms that are syntactic representations of (finite) computation trees.

A remarkable result from the meta-theory of SOS is offered by [5]. There the authors provide an algorithm capable of extracting a sound and ground complete set of axioms for bisimilarity from a TSS in the GSOS format. The method of [5] has been later adapted in [32] in order to address GSOS languages with an explicit notion of successful termination encoded as a predicate symbol, and recently, in [6], to address GSOS languages where predicates can be defined in a general fashion.

It is not possible to do justice to all the type of results that the literature provides on the meta-theory of SOS and the list above must be understood as a personal choice of the most relevant streams of research. The reader may also be interested in considering ordered SOS [143, 106, 105], decompositon of modal logics [88, 58] and time [82, 144], probability [35, 83] and security [104, 142, 141] related meta-theorems, just to mention some. Excellent overviews of the results in this field, although a little dated, are [18] and [109].

For the sake of concreteness, we limit ourselves to presenting two examples of meta-results.

Example 1.3.1 (An example of congruence format) *Consider the GSOS format of Bloom, Istrail and Meyer [44], whose definition is given below.*

Definition 1.3.2 (GSOS rule) *Suppose Σ is a signature. A* GSOS rule *r over Σ is a rule of the form:*

$$\frac{\bigcup_{i=1}^{l}\left\{x_i \xrightarrow{a_{ij}} y_{ij} | 1 \leq j \leq m_i\right\} \quad \cup \quad \bigcup_{i=1}^{l}\left\{x_i \overset{b_{ik}}{\nrightarrow} | 1 \leq k \leq n_i\right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} t}$$

where all the variables are distinct, $m_i, n_i \geq 0$, a_{ij}, b_{ik}, and c are actions from a finite set, f is a function symbol from Σ with arity l, and t is a term that may only contain variables in the set $\{x_1, \ldots, x_l\} \cup \{y_{ij} \mid 1 \leq i \leq l, 1 \leq j \leq m_i\}$.

Definition 1.3.3 *A* GSOS language *is a triple $G = (\Sigma_G, \mathcal{L}, R_G)$, where Σ_G is a finite signature, $\mathcal{L}$ is a finite set of action labels and R_G is a finite set of GSOS rules over Σ_G.*

Denoting with the symbol $\leftrightarrows$ the bisimilarity, the following theorem is a classic result in the meta-theory of SOS, [44].

Theorem 1.3.1 *For every GSOS language $\leftrightarrows$ is a congruence.*

Proving bisimilarity to be a congruence may be sometimes a tricky task and this kind of results is indeed a precious contribution since it also saves researchers some work. For

example, the reader should notice that the semantics for the simple concurrent language given in Example 1.2.1 fits the GSOS format. This is enough to conclude that bisimilarity is a congruence for that language, without embarking on any specific proof.

Example 1.3.4 (A rule format for idempotence) *The reader might also be pleased to see an example of rule format that is capable of ensuring the validity of some algebraic law. We now proceed to present a result extracted from [4] that guarantees that certain binary operators are idempotent w.r.t to bisimilarity, i.e. provided f is a binary function symbol, it holds that $f(x,x) \leftrightarrow x$.*

Definition 1.3.5 *A TSS is in the idempotence format with respect to an operator f whenever each f-defining rule is of the form*

$$\frac{\{x_i \xrightarrow{l} t\} \cup \Phi}{f(x_0, x_1) \xrightarrow{l} t}, \quad i \in \{0,1\}$$

where Φ is an arbitrary set of premises and it holds that there exists at least one rule for f for which Φ is empty.

In [4] the authors consider other kinds of rules capable of ensuring the idempotence of operators; since they are not strictly relevant in our context, they are omitted here for the sake of simplicity. The interested reader is invited to read the original source for the description of the general format. The following theorem is a simplified version of the result in [4].

Theorem 1.3.2 *The equation $f(x,x) \leftrightarrow x$ holds in any complete TSS in the idempotence format with respect to a binary operator f.*

As a matter of fact, the TSS of the Example 1.2.1 is in the idempotence format with respect to $+$. This is enough to conclude that $+$ is idempotent, saving us from the annoying task to provide an ad hoc proof of that fact.

Why the meta-theory of SOS is so appealing Results in the meta-theory of SOS are usually very general and broadly applicable. It is hard to summarize all the benefits ensuing from the results of the meta-theory of SOS developed so far. In my opinion, the most important ones are as follows.

- These type of results are desirable because they do not apply only to one particular language, but they instead apply to many languages, namely every

language whose semantic description can be defined within the restricted format considered.

- Most of the results in the meta-theory of SOS allow the user to prove some semantic property by performing a simple syntactic check based on the form of the rules employed in writing the semantics of the language at hand. Semantic properties like the congruence of some equivalence relation or the validity of some algebraic law are two typical examples of properties that a user may want to prove about a language. These properties may be long and tedious to prove, may be just routine work, or sometimes even tricky. Rule formats save the user from embarking on a proof which is ad-hoc for the language at hand: it is sufficient to show that some syntactic restrictions are met and the semantic property is then guaranteed to hold for free.
- Last but not least, studying meta-theory is a way to discover insights in programming languages and their semantics, i.e. it is a way to understand, at a syntactic level, why certain languages afford a property while others do not.

Moreover, being the meta-theory of SOS a mature and successful field by now, I personally believe that the wealth of useful existing contributions represents further reason to use SOS in the first place, possibly increasing the already large appeal of SOS when it comes to developing and analyzing new languages.

1.4 Contributions: A summary

During my Ph.D. thesis work, I offered with my colleagues a number of contributions to the meta-theory of SOS. The contributions presented in my dissertation are not specific to a single subject. Rather, I worked on a few different streams of research.

This section contains a brief survey of the results presented in the thesis and indicate the papers on which they are based. It gives also an overview of the structure of the thesis. All but one of the papers have been published in refereed conference proceedings or journals.

- **Proving equivalence of open terms**. Chapter 2 is devoted to the study of a bisimulation-based method for establishing the soundness of equations between terms constructed using operations whose semantics is specified

by rules in the GSOS format of Bloom, Istrail and Meyer, [44]. The method is inspired by de Simone's FH-bisimilarity, [52], and uses transition rules as schematic transitions in a bisimulation-like relation between open terms. The soundness of the method is proven and examples showing its applicability are provided. The proposed bisimulation-based proof method is incomplete, but the chapter offers some completeness results for restricted classes of GSOS specifications. An extension of the proof method to the setting of GSOS languages with predicates is also offered. The material contained in this chapter is based on the content of the published papers [7] and [8].

- **Rule formats for zero and unit elements**. Chapter 3 is devoted to the study of rule formats for SOS guaranteeing that certain constants act as left or right zero elements for a set of binary operators. Our design approach is also applied to reformulate an earlier rule format for unit elements. Examples of left and right zero, as well as unit, elements from the literature are shown to be checkable using the provided formats. The material contained in this chapter is based on the content of the published papers [11] and [9].

- **Rule formats for distributivity**. Chapter 4 is devoted to the presentation of rule formats for SOS guaranteeing that certain binary operators are left distributive with respect to a set of binary operators. Examples of left-distributivity laws from the literature are shown to be instances of the provided formats. Some conditions ensuring the impossibility of the validity of the left-distributivity law are also offered. The material contained in this chapter is based on the content of the published paper [12] and the technical report [10], a version of which has been submitted for journal publication.

- **Structural operational semantics with binders**. Chapter 5 is devoted to the development of a new formal framework for accomodating binders and names in SOS. The framework is based on the Nominal Logic of Gabbay and Pitts and hence is called Nominal SOS. We formulate the lazy λ-calculus, [2], and the early π-calculus, [132, 98], in our framework and we prove the correctness of these calculi w.r.t. their original operational semantics. We define a notion of nominal bisimilarity and show that for the early π-calculus this notion is fully abstract w.r.t. open bisimilarity.

  The content of Chapter 5 is based on the early ideas by Murdoch James Gabbay, MohammadReza Mousavi and Michel Reniers. Substantial further developments have been carried out by me in a joint collaboration with them.

This chapter presents the results from the unpublished material grown from this collaboration.

Chapters 2-5 are devoted to different contributions and each of the chapters is self-contained. The reader that is interested in reading one particular contribution can skip directly to the corresponding chapter.

Chapter 6 concludes the thesis with a discussion of its main contributions and points out some directions for future work.

1.4.1 Publications resulting from the thesis work

We list below the publications resulted from my 3-year long Ph.D. thesis work.

1. Luca Aceto, Matteo Cimini, and Anna Ingólfsdóttir. A bisimulation-based method for proving the validity of equations in GSOS languages. *In Proceedings of the 6th Workshop on Structural Operational Semantics 2009 (SOS 2009), August 31, 2009, Bologna (Italy), volume 18 of Electronic Proceedings in Theoretical Computer Science*, pages 1-16, 2010.

2. Luca Aceto, Matteo Cimini, and Anna Ingólfsdóttir. Proving the validity of equations in GSOS languages using rule-matching bisimilarity, 2011. to appear in *Mathematical Structures in Computer Science.*

3. Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, MohammadReza Mousavi, and Michel A. Reniers. SOS rule formats for zero and unit elements. *Theoretical Computer Science, 412(28):3045-3071, 2011.*

4. Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, MohammadReza Mousavi, and Michel A. Reniers. On rule formats for zero and unit elements. *In Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVI), Ottawa, Canada, volume 265 of Electronic Notes in Theoretical Computer Science*, pages 145-160. Elsevier B.V., The Netherlands, 2010.

5. Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Mohammadreza Mousavi, and Michel A. Reniers. Rule formats for distributivity. *In Proceedings of the 5th International Conference on Language and Automata Theory and Applications (LATA 2011), volume 6638 of Lecture Notes in Computer Science,* pages 79-90, Springer-Verlag, 2011.

6. Matteo Cimini, Claudio Sacerdoti Coen, and Davide Sangiorgi. Functions as processes: Termination and the $\bar{\lambda}\mu\tilde{\mu}$-Calculus. *In Proceedings of the 5th Symposium on Trustworthy Global Computing (TGC 2010), volume 6084 of Lecture Notes in Computer Science,* pages 73-86, Springer-Verlag, 2010.

7. Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Arni Hermann Reynisson, Steinar Hugi Sigurdarson, and Marjan Sirjani. Modelling and simulation of asynchronous real-time systems using Timed Rebeca. *In Proceedings of the 10th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2011), volume 58 of Electronic Proceedings in Theoretical Computer Science*, pages 1-19, 2011.

As remarked in the previous section, none of the papers above concerns the contribution presented in Chapter 5. The content of Chapter 5 is based on unpublished material reporting on joint work with Murdoch James Gabbay, MohammadReza Mousavi and Michel Reniers that we plan to submit for publication soon.

Also, contributions 6 and 7 concern SOS in a more applicative fashion and they do not explicitly deal with the meta-theory of SOS. The results from those papers are therefore not part of the present thesis.

Chapter 2

Proving Equivalence of Open Terms

All animals are equal, but some animals are more equal than others.
George Orwell, from the book *Animal Farm*.

2.1 Introduction

Equations play a fundamental role in the development of the theory and practice of process calculi and programming languages since they offer a mathematically appealing and concise way of stating the 'laws of programming' (to borrow the title of a paper by Hoare et al. [78]) that apply to the language at hand. In the setting of process calculi, the study of equational axiomatizations of behavioural relations has been a classic area of investigation since, *e.g.*, the early work of Hennessy and Milner [74, 95], who offered complete axiom systems for bisimilarity [115] over the finite and regular fragments of Milner's CCS [96]. Such axiomatizations capture the essence of bisimilarity over those fragments of CCS in a syntactic, and often revealing, way and pave the way for the verification of equivalences between processes by means of theorem proving techniques. Despite these early achievements, the search for axiomatizations of process equivalences that are powerful enough to establish all the valid equations between *open* process terms (that is, terms possibly containing variables) has proven to be a very difficult research problem; see [14] for a survey of results in this area. For instance, to the best of our knowledge, there is no known axiomatization of bisimilarity over

recursion-free CCS that is complete over open terms. Stepping stones towards such a result are offered in, *e.g.*, [15, 20].

The most basic property of any equation is that it be *sound* with respect to the chosen notion of semantics. Soundness proofs are often lengthy, work-intensive and need to be carried out for many equations and languages. It is therefore not surprising that the development of general methods for proving equivalences between open terms in expressive process calculi has received some attention since the early developments of the algebraic theory of processes—see, *e.g.*, the references [47, 88, 125, 52, 149] for some of the work in this area over a period of over 20 years. This article offers a contribution to this line of research by developing a bisimulation-based method, which we call *rule-matching bisimilarity*, for establishing the soundness of equations between terms constructed using operations whose semantics is specified by rules in the GSOS format of Bloom, Istrail and Meyer [44]. Rule-matching bisimilarity is inspired by de Simone's FH-bisimilarity [52] and uses transition rules as transition schemas in a bisimulation-like relation between open terms. We prove that rule-matching bisimilarity is a sound proof method for showing the validity of equations with respect to bisimilarity and exhibit examples witnessing its incompleteness.

The incompleteness of rule-matching bisimilarity is not unexpected and raises the question whether the method is powerful enough to prove the soundness of 'interesting' equations. In order to offer a partial answer to this question, we provide examples showing the applicability of our proof method. In particular, our method does not only apply to a more expressive rule format than the one proposed by de Simone in [52], but is also a sharpening of de Simone's FH-bisimilarity over de Simone languages. See Section 2.6, where we apply rule-matching bisimilarity to prove the soundness of the equations in de Simone's 'clock example'. (This example was discussed by de Simone in [52] to highlight the incompleteness of FH-bisimilarity.) On the theoretical side, we also offer some completeness results for restricted classes of GSOS specifications.

We also extend our main results to the setting of GSOS language specifications with predicates. This extension, albeit not theoretically deep, is significant from the point of view of applications of rule-matching bisimilarity since the operational semantics of several operations commonly found in the literature on, *e.g.*, process algebra is best specified using rules involving the use of predicates as first-class notions.

Overall, we believe that, while our conditions are neither necessary nor in general can they be checked algorithmically, they frequently hold, and they are more accessible to machine support than a direct proof of soundness.

The chapter is organized as follows. Sections 2.2 and 2.3 introduce the necessary preliminaries on the GSOS rule format that are needed in the remainder of the chapter. In particular, Section 2.3 recalls the notion of ruloid, which plays a key role in the technical developments to follow. In Section 2.4, we introduce a simple logic of transition formulae and establish a decidability result for the validity of implications between formulae. Implication between certain kinds of transition formulae that are naturally associated with the premises of (sets of) ruloids is used in the definition of rule-matching bisimilarity in Section 2.5. In that section, we prove that rule-matching bisimilarity is a sound method for showing the validity of equations in GSOS languages modulo bisimilarity and exhibit examples witnessing its incompleteness. We apply rule-matching bisimilarity to show the validity of some sample equations from the literature on process algebra in Section 2.6. We then offer some partial completeness results for rule-matching bisimilarity (Section 2.7). Section 2.8 is devoted to the extension of our main results to the setting of GSOS languages with predicates. The chapter concludes with a discussion of related and future work (Section 2.9). For the sake of readability, proofs of some technical results are collected in a series of sections that follow Section 2.9.

2.2 Preliminaries

We assume familiarity with the basic notation of process algebra and structural operational semantics; see *e.g.* [18, 24, 44, 68, 72, 77, 96, 109, 65] for more details.

Let Var be a countably infinite set of *process variables* with typical elements x, y. A *signature* Σ consists of a set of *operation symbols*, disjoint from Var, together with a function *arity* that assigns a natural number to each operation symbol. The set $\mathbb{T}(\Sigma)$ of *terms* built from the operations in Σ and the variables in Var is the least set such that

- each $x \in \mathsf{Var}$ is a term;
- if f is an operation symbol of arity l, and $P_1, \ldots, P_l$ are terms, then $f(P_1, \ldots, P_l)$ is a term.

We use $P, Q, \ldots$ to range over terms and the symbol $\equiv$ for the relation of syntactic equality on terms. We denote by $\mathrm{T}(\Sigma)$ the set of *closed* terms over Σ, *i.e.*, terms that do not contain variables, and will use $p, q, \ldots$ to range over it. An operation symbol f of arity 0 will be often called a *constant* symbol, and the term $f()$ will be abbreviated as f.

Besides terms we have *actions*, elements of some given nonempty, finite set Act, which is ranged over by a, b, c, d. A *positive transition formula* is a triple of two terms and an action, written $P \xrightarrow{a} P'$. A *negative transition formula* is a pair of a term and an action, written $P \overset{a}{\nrightarrow}$.

A *(closed)* Σ*-substitution* is a function σ from variables to (closed) terms over the signature Σ. For t a term or a transition formula, we write $t\sigma$ for the result of substituting $\sigma(x)$ for each x occurring in t, and $vars(t)$ for the set of variables occurring in t. A Σ*-context* $C[\vec{x}]$ is a term in which at most the variables $\vec{x}$ appear. $C[\vec{P}]$ is $C[\vec{x}]$ with x_i replaced by P_i wherever it occurs.

Definition 2.2.1 (GSOS rule) *Suppose Σ is a signature. A* GSOS rule *ρ over Σ is a rule of the form:*

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \le j \le m_i \right\} \;\cup\; \bigcup_{i=1}^{l} \left\{ x_i \overset{b_{ik}}{\nrightarrow} \mid 1 \le k \le n_i \right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]} \tag{2.1}$$

where all the variables are distinct, $m_i, n_i \ge 0$, a_{ij}, b_{ik}, and c are actions, f is an operation symbol from Σ with arity l, and $C[\vec{x}, \vec{y}]$ is a Σ-context.

It is useful to name components of rules. The operation symbol f is the principal operation *of the rule, and the term $f(\vec{x})$ is the* source*. $C[\vec{x}, \vec{y}]$ is the* target*; c is the* action*; the formulae above the line are the* antecedents *(sometimes denoted by $\mathsf{ante}(\rho)$); and the formula below the line is the* consequent *(sometimes denoted by $\mathsf{cons}(\rho)$).*

For a GSOS rule ρ, we use $\mathsf{SV}(\rho)$ and $\mathsf{TV}(\rho)$ to denote the sets of source and target variables of ρ, respectively; that is, $\mathsf{SV}(\rho)$ is the set of variables in the source of ρ, and $\mathsf{TV}(\rho)$ is the set of y's for antecedents $x \xrightarrow{a} y$.

Definition 2.2.2 *A* GSOS language *is a pair $G = (\Sigma_G, R_G)$ where Σ_G is a finite signature and R_G is a finite set of GSOS rules over Σ_G.*

Informally, the intent of a GSOS rule is as follows. Suppose that we are wondering whether $f(\vec{P})$ is capable of taking a c-step. We look at each rule with principal operation f and action c in turn. We inspect each positive antecedent $x_i \xrightarrow{a_{ij}} y_{ij}$,

checking if P_i is capable of taking an a_{ij}-step for each j and if so calling the a_{ij}-children Q_{ij}. We also check the negative antecedents; if P_i is *in*capable of taking a b_{ik}-step for each k. If so, then the rule *fires* and $f(\vec{P}) \xrightarrow{c} C[\vec{P}, \vec{Q}]$. This means that the transition relation $\rightarrow_G$ associated with a GSOS language G is the one defined by the rules using structural induction over closed Σ_G-terms.

For the sake of precision, we will now formally define the transition relation induced by a GSOS language.

Definition 2.2.3 *A* transition relation *over a signature Σ is a relation $\rightsquigarrow \subseteq \mathrm{T}(\Sigma) \times \mathsf{Act} \times \mathrm{T}(\Sigma)$. We write $p \overset{a}{\rightsquigarrow} q$ as an abbreviation for $(p, a, q) \in \rightsquigarrow$.*

Definition 2.2.4 *Suppose $\rightsquigarrow$ is a transition relation and σ a closed substitution. For each transition formula ϕ, the predicate $\rightsquigarrow, \sigma \models \phi$ is defined by*

$$\begin{aligned} \rightsquigarrow, \sigma \models P \xrightarrow{a} Q &\triangleq P\sigma \overset{a}{\rightsquigarrow} Q\sigma \\ \rightsquigarrow, \sigma \models P \overset{a}{\nrightarrow} &\triangleq \nexists Q : P\sigma \overset{a}{\rightsquigarrow} Q \end{aligned}$$

For H a set of transition formulae, we define

$$\rightsquigarrow, \sigma \models H \triangleq \forall \phi \in H : \ \rightsquigarrow, \sigma \models \phi$$

and for $\dfrac{H}{\phi}$ a GSOS rule,

$$\rightsquigarrow, \sigma \models \frac{H}{\phi} \triangleq \Big(\rightsquigarrow, \sigma \models H \implies \rightsquigarrow, \sigma \models \phi\Big).$$

For t a transition formula, a set of such formulae or a GSOS rule, we sometimes abbreviate $\rightsquigarrow, \sigma \models t$ to $\sigma \models t$ when the transition relation is clear from the context.

Definition 2.2.5 *Suppose G is a GSOS language and $\rightsquigarrow$ is a transition relation over Σ_G. Then $\rightsquigarrow$ is* sound *for G iff for every rule $\rho \in R_G$ and every closed Σ_G-substitution σ, we have $\rightsquigarrow, \sigma \models \rho$. A transition $p \overset{a}{\rightsquigarrow} q$ is* supported *by some rule $\dfrac{H}{\phi} \in R_G$ iff there exists a substitution σ such that $\rightsquigarrow, \sigma \models H$ and $\phi\sigma = \Big(p \xrightarrow{a} q\Big)$. The relation $\rightsquigarrow$ is* supported *by G iff each transition in $\rightsquigarrow$ is supported by a rule in R_G.*

It is well known that the requirements of soundness and supportedness are sufficient to associate a unique transition relation with each GSOS language.

Lemma 2.2.6 ([44]) *For each GSOS language G there is a unique sound and supported transition relation.*

We write $\rightarrow_G$ for the unique sound and supported transition relation for G. We say that a rule ρ is *junk* in G if it does not support any transition of $\rightarrow_G$. For each closed term p, we define $init(p) = \{a \in \mathsf{Act} \mid \exists q : \ p \xrightarrow{a}_G q\}$. For a GSOS language G, we let $init(\mathrm{T}(\Sigma_G)) = \{init(p) \mid p \in \mathrm{T}(\Sigma_G)\}$.

The basic notion of equivalence among terms of a GSOS language we will consider in this work is *bisimulation equivalence* [96, 115].

Definition 2.2.7 *Suppose G is a GSOS language. A binary relation $\sim \ \subseteq \mathrm{T}(\Sigma_G) \times \mathrm{T}(\Sigma_G)$ over closed terms is a* bisimulation *if it is symmetric and $p \sim q$ implies, for all $a \in \mathsf{Act}$,*

If $p \xrightarrow{a}_G p'$ then, for some q', $q \xrightarrow{a}_G q'$ and $p' \sim q'$.

We write $p \leftrightarrows_G q$ if there exists a bisimulation $\sim$ relating p and q. The subscript G is omitted when it is clear from the context.

It is well known that $\leftrightarrows_G$ is a congruence for all operation symbols f of G [44].

Let $\mathsf{Bisim}(G)$ denote the quotient algebra of closed Σ_G-terms modulo bisimulation. Then, for $P, Q \in \mathbb{T}(\Sigma_G)$,

$$\mathsf{Bisim}(G) \models P = Q \ \Leftrightarrow \ (\forall \text{ closed } \Sigma_G\text{-substitutions } \sigma : P\sigma \leftrightarrows_G Q\sigma).$$

In what follows, we shall sometimes consider equations that hold over all GSOS languages that extend a GSOS language G with new operation symbols and rules for the new operations. The following notions from [5] put these extensions on a formal footing.

Definition 2.2.8 *A GSOS language G' is a* disjoint extension *of a GSOS language G if the signature and rules of G' include those of G, and G' introduces no new rules for operations of G.*

If G' disjointly extends G then G' introduces no new outgoing transitions for the closed terms of G. This means in particular that $P \leftrightarrows_G Q$ iff $P \leftrightarrows_{G'} Q$, for $P, Q \in \mathrm{T}(\Sigma_G)$. (More general conservative extension results are discussed in, *e.g.*, [59, 107].)

For G a GSOS language, let $\mathsf{BISIM}(G)$ stand for the class of all algebras $\mathsf{Bisim}(G')$, for G' a disjoint extension of G. Thus we have, for $P, Q \in \mathbb{T}(\Sigma_G)$,

$$\mathsf{BISIM}(G) \models P = Q \;\Leftrightarrow\; (\forall G' : G' \text{ a disjoint extension of } G \implies \mathsf{Bisim}(G') \models P = Q).$$

Checking the validity of a statement of the form $\mathsf{Bisim}(G) \models P = Q$ or $\mathsf{BISIM}(G) \models P = Q$ according to the above definition is at best very impractical, as it involves establishing bisimilarity of all closed instantiations of the terms P and Q. It would thus be helpful to have techniques that use only information obtainable from these terms and that can be used to this end. The development of one such technique will be the subject of the remainder of this chaprter.

2.2.1 Eliminating junk rules

Note that the definition of a GSOS language given above does not exclude *junk* rules, *i.e.*, rules that support no transition in $\rightarrow_G$. For example, the rule

$$\frac{x \xrightarrow{a} y, \qquad x \overset{a}{\nrightarrow}}{f(x) \xrightarrow{a} f(y)}$$

has contradictory antecedents and can never fire. Also it can be the case that a (seemingly innocuous) rule like

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{b} f(y)}$$

does not support any transition if $\rightarrow_G$ contains no a-transitions. The possible presence of junk rules does not create any problems in the development of the theory of GSOS languages as presented in [5, 44] and the authors of those papers saw no reason to deal with these rules explicitly.

Our aim in this work is to develop a test for the validity of equalities between open terms in GSOS languages. The test we shall present in later sections is based upon the idea of using GSOS rules as 'abstract transitions' in a bisimulation-like equivalence between open terms. In order to ease the applicability of this method, it is thus desirable, albeit not strictly necessary (see Remark 2.5.2), to eliminate junk rules from GSOS languages, as these rules would be interpreted as 'potential transitions' from a term which, however, cannot be realized.

Consider, for example, the trivial GSOS language TRIV with unary operations f and g, and rule

$$f(x) \xrightarrow{a} f(x) \ .$$

It is immediate to see that $\mathsf{Bisim}(\mathrm{TRIV}) \models f(x) = g(y)$ as the set of closed terms in TRIV is empty. However, if we considered the rule for f as a transition from $f(x)$ in a simple-minded way, we would be led to distinguish $f(x)$ and $g(y)$ as the former has a transition while the latter does not. Obviously, the rule for f given above is junk.

Clearly junk rules can be removed from a GSOS language G without altering the associated transition relation. Of course, in order to be able to remove junk rules from a GSOS language, we need to be able to discover effectively what rules are junk. This is indeed possible, as the following theorem, due to Aceto, Bloom and Vaandrager [18, Theorem 5.22], shows. (Below we present a proof of this result since we refer to it in the proofs of Theorems 2.4.1 and 2.8.7.)

Theorem 2.2.9 *Let $G = (\Sigma_G, R_G)$ be a GSOS language. Suppose that $\rho \in R_G$. Then it is decidable whether ρ is junk in G.*

Proof. Let $G = (\Sigma_G, R_G)$ be a GSOS language. First of all, note that it is easy to determine which rules in R_G are junk once we have computed the set

$$init(\mathrm{T}(\Sigma_G)) \quad = \quad \{init(p) \mid p \in \mathrm{T}(\Sigma_G)\}$$

where $init(p) = \{a \in \mathsf{Act} \mid \exists q : \ p \xrightarrow{a} q\}$. In fact, it is immediate to see that the hypotheses of a GSOS rule of the form (2.1) are satisfiable iff there exist processes $p_1, \ldots, p_l \in \mathrm{T}(\Sigma_G)$ such that $\left\{a_{ij} | 1 \leq j \leq m_i\right\} \subseteq init(p_i)$ and $\{b_{ik} | 1 \leq k \leq n_i\} \cap init(p_i) = \emptyset$ for all $1 \leq i \leq l$.

So we are left to give an effective way of computing the set $init(\mathrm{T}(\Sigma_G))$ for any GSOS language G. This we do as follows. First of all, note that each function symbol $f \in \Sigma_G$ of arity l determines a computable function

$$\hat{f} : \underbrace{2^{\mathsf{Act}} \times \cdots \times 2^{\mathsf{Act}}}_{l\text{-times}} \to 2^{\mathsf{Act}}$$

by $\hat{f}(X_1, \ldots, X_l) = Y$, where for all $c \in \mathsf{Act}$, $c \in Y$ iff there exists a rule ρ for f of the form (2.1) with action c such that, for all $1 \leq i \leq l$, $\left\{a_{ij} | 1 \leq j \leq m_i\right\} \subseteq X_i$ and $\{b_{ik} | 1 \leq k \leq n_i\} \cap X_i = \emptyset$. (If f is a constant, then we use $\hat{f}$ to denote the subset Y of

Act such that $f(\{\bullet\}) = Y$.) Now, for each $X \subseteq 2^{\mathsf{Act}}$, let $\mathcal{G}(X)$ be given by

$$\mathcal{G}(X) \triangleq \{Y \mid \exists f \in \Sigma_G,\ X_1, \ldots, X_l \in X : \hat{f}(X_1, \ldots, X_l) = Y\}.$$

Note that, for each $X \subseteq 2^{\mathsf{Act}}$, $\mathcal{G}(X)$ can be effectively computed and that $X \subseteq Y$ implies $\mathcal{G}(X) \subseteq \mathcal{G}(Y)$.

Our strategy for computing $\mathit{init}(\mathrm{T}(\Sigma_G))$ is to divide the set of closed terms $\mathrm{T}(\Sigma_G)$ into sets T_i of terms with depth less than or equal to i, and to compute the nondecreasing sequence

$$\mathit{init}(T_1) \subseteq \mathit{init}(T_2) \subseteq \cdots$$

until it stabilizes. Obviously, this sequence will stabilize in a finite number of steps as Act is finite.

Now, set T_1 contains all the constants in the language. Thus $\mathit{init}(T_1)$ can be computed by inspection of the rules for the constants. (Note that, by the form of the rules, these have no antecedents.) In fact, we have that $\mathit{init}(T_1) = \left\{\hat{f} \mid f \text{ a constant symbol in } \Sigma_G\right\}$. So suppose that we want to compute $\mathit{init}(T_{i+1})$ given that we already have $\mathit{init}(T_i)$. We claim that $\mathit{init}(T_{i+1}) = \mathcal{G}(\mathit{init}(T_i))$. In fact, each term in T_{i+1} is of the form $f(p_1, \ldots, p_l)$, where the p_i's are all in T_i. Thus we know $\mathit{init}(p_i)$ for all $1 \leq i \leq l$ and that is exactly what we need in order to determine which rules for f can fire from that term. Hence we can compute $\mathit{init}(T_1)$, and each $\mathit{init}(T_{i+1})$ can be computed from $\mathit{init}(T_i)$ using the monotonic and effective operation $\mathcal{G}(\cdot)$. This completes the proof.

Note that a GSOS rule without antecedents, *i.e.* an axiom, is junk iff the set of closed terms is empty. For example, as mentioned before, the rule for the operation f in TRIV is junk.

As a further example, consider the GSOS language G with constant a^ω and unary operation f with rules

$$\frac{}{a^\omega \xrightarrow{a} a^\omega} \qquad \frac{}{f(x) \xrightarrow{a} f(x)} \qquad \frac{x \xrightarrow{b} y}{f(x) \xrightarrow{b} f(y)}$$

Using our decision procedure, it is immediate to check that $\mathit{init}(\mathrm{T}(\Sigma_G)) = \{\{a\}\}$. Thus the rule

$$\frac{x \xrightarrow{b} y}{f(x) \xrightarrow{b} f(y)}$$

is junk in G, as its antecedent cannot be satisfied.

As a consequence of the above theorem, all the junk rules in a GSOS language can be effectively removed in a pre-processing step before applying the techniques described in the subsequent sections. Thus we will henceforth restrict ourselves to GSOS languages without junk rules.

2.3 Ruloids and the operational specification of contexts

As mentioned above, the essence of our method for checking the validity of equations in GSOS languages is to devise a variation on bisimulation equivalence between contexts that considers GSOS rules as transitions. For primitive operations in a GSOS language G, the rules in R_G will be viewed as abstract transitions from terms of the form $f(\vec{x})$. However, in general, we will be dealing with complex contexts in $\mathbb{T}(\Sigma_G)$. In order to apply our ideas to general open terms, we will thus need to associate with arbitrary contexts a set of derived rules (referred to as *ruloids* [44]) describing their behaviour.

A *ruloid* for a context $D[\vec{x}]$, with $\vec{x} = (x_1, \ldots, x_l)$, takes the form:

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq j \leq m_i \right\} \quad \cup \quad \bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{b_{ik}\!\!\!\!\rightarrow} \mid 1 \leq k \leq n_i \right\}}{D[\vec{x}] \xrightarrow{c} C[\vec{x}, \vec{y}]} \tag{2.2}$$

where the variables are distinct, $m_i, n_i \geq 0$, a_{ij}, b_{ik}, and c are actions, and $C[\vec{x}, \vec{y}]$ is a Σ-context.

A ruloid ρ with the above form is sound for $D[\vec{x}]$ iff for every closed Σ_G-substitution σ, we have $\rightarrow_G, \sigma \models \rho$. A set of such ruloids is sound for $D[\vec{x}]$ if so is each of its members.

Definition 2.3.1 *A set of ruloids R is* supporting[1] *for a context $D[\vec{x}]$ and action c iff all the consequents of ruloids in R are of the form $D[\vec{x}] \xrightarrow{c} C[\vec{x}, \vec{y}]$ and, whenever $D[\vec{P}] \xrightarrow{c}_G p$, there are a ruloid $\rho \in R$ and a closed substitution σ such that $\mathsf{cons}(\rho)\sigma = D[\vec{P}] \xrightarrow{c} p$ and $\rightarrow_G, \sigma \models \mathsf{ante}(\rho)$.*

[1] Our terminology departs slightly from that of [44]. Bloom, Istrail and Meyer use 'specifically witnessing' in lieu of 'supporting'.

The following theorem is a slightly sharpened version of the Ruloid Theorem (Theorem 7.4.3) in [44].

Theorem 2.3.2 (Ruloid theorem) *Let G be a GSOS language and $X \subseteq \mathsf{Var}$ be a finite set of variables. For each $D[\vec{x}] \in \mathbb{T}(\Sigma_G)$ and action c, there exists a finite set $R_{D,c}$ of ruloids of the form (2.2) such that:*

1. *$R_{D,c}$ is sound and supporting for $D[\vec{x}]$, and*
2. *$\mathsf{TV}(\rho) \cap X = \emptyset$, for every $\rho \in R_{D,c}$.*

Moreover, the set $R_{D,c}$ can be effectively constructed.

Proof. A straightforward adaptation of the proof of the corresponding result in [44], where we take care in choosing the target variables in ruloids so that condition 2 in the statement of the theorem is met.

Definition 2.3.3 *Let G be a GSOS language. For each $D[\vec{x}] \in \mathbb{T}(\Sigma_G)$, the ruloid set of $D[\vec{x}]$, notation $R_G\big(D[\vec{x}]\big)$, is the union of the sets $R_{D,c}$ ($c \in \mathsf{Act}$) given by Theorem 2.3.2.*

Remark 2.3.4 *A set of ruloids that is supporting for a context $D[\vec{x}]$ and action a may have size that is exponential in the number of variables in $\vec{x}$. By way of example, consider the context $D[x_1, \ldots, x_n] = f(g(x_1), \ldots, g(x_n))$, where the rules for f and g are as follows.*

$$\frac{\{x_i \xrightarrow{a} y_i \mid 1 \leq i \leq n\}}{f(x_1, \ldots, x_n) \xrightarrow{a} f(x_1, \ldots, x_n)} \qquad \frac{x \xrightarrow{a} y}{g(x) \xrightarrow{a} y} \qquad \frac{x \xrightarrow{b} y}{g(x) \xrightarrow{a} g(y)}$$

It is easy to see that there are 2^n ruloids for $D[x_1, \ldots, x_n]$ with action a.

The import of the Ruloid Theorem is that the operational semantics of an open term P can be described by a finite set $R_G(P)$ of derived GSOS-like rules. Examples of versions of the above result for more expressive formats of operational rules may be found in, *e.g.*, the references [43, 58].

Example 2.3.5 *Consider a GSOS language G containing the sequencing operation ';' specified by the following rules (one such pair of rules for each $a \in \mathsf{Act}$).*

$$\frac{x \xrightarrow{a} z}{x; y \xrightarrow{a} z; y} \qquad \frac{x \xrightarrow{b} \not\to \ (\forall b \in \mathsf{Act}),\ y \xrightarrow{a} z}{x; y \xrightarrow{a} z} \tag{2.3}$$

Let $R[x, y, z] = x; (y; z)$ and $L[x, y, z] = (x; y); z$. The ruloids for L and R are:

$$\frac{x \xrightarrow{a} x'}{\begin{array}{c} L \xrightarrow{a} (x';y);z \\ R \xrightarrow{a} x';(y;z) \end{array}} \qquad \frac{x \not\xrightarrow{}, y \xrightarrow{a} y'}{\begin{array}{c} L \xrightarrow{a} y';z \\ R \xrightarrow{a} y';z \end{array}} \qquad \frac{x \not\xrightarrow{}, y \not\xrightarrow{}, z \xrightarrow{a} z'}{\begin{array}{c} L \xrightarrow{a} z' \\ R \xrightarrow{a} z' \end{array}} \tag{2.4}$$

where we write $x \not\xrightarrow{}$ in the antecedents of ruloids as a shorthand for $x \not\xrightarrow{b}$ $(\forall b \in \mathsf{Act})$.

Remark 2.3.6 *Note that the set $R_G\big(D[\vec{x}]\big)$ of ruloids for a context $D[\vec{x}]$ in a GSOS language G may be chosen to contain junk ruloids even when G has no junk rule. For example, consider the GSOS language with constants a and $\mathbf{0}$, unary operation g and binary operation f with the following rules.*

$$\frac{}{a \xrightarrow{a} \mathbf{0}} \qquad \frac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{f(x,y) \xrightarrow{a} \mathbf{0}} \qquad \frac{x \not\xrightarrow{a}}{g(x) \xrightarrow{b} \mathbf{0}}$$

None of the above rules is junk. However, the only ruloid for the context $f(x, g(x))$ is

$$\frac{x \xrightarrow{a} x', x \not\xrightarrow{a}}{f(x,g(x)) \xrightarrow{a} \mathbf{0}} \ ,$$

which is junk. However, junk ruloids can be removed from the set of ruloids for a context using Theorem 2.2.9. In what follows, we shall assume that the set of ruloids we consider have no junk ruloids.

In the standard theory on GSOS, it was not necessary to pay much attention to the variables in rules and ruloids, as one was only interested in the transition relation they induced over *closed terms*. (In the terminology of [68], all the variables occurring in a GSOS rule/ruloid are not *free*.) Here, however, we intend to use ruloids as abstract transitions between open terms. In this framework it becomes desirable to give a more reasoned account of the role played by variables in ruloids, as the following example shows.

Example 2.3.7 *Consider a GSOS language G containing the unary operations f and g with the following rules.*

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} y} \qquad \frac{x \xrightarrow{a} z}{g(x) \xrightarrow{a} z}$$

It is easy to see that $\mathsf{Bisim}(G) \models f(x) = g(x)$, regardless of the precise description of G. However, in order to prove this equality, any bisimulation-like equivalence relating open terms in $\mathbb{T}(\Sigma_G)$ would have to relate the variables y and z in some way. Of course, this

will have to be done carefully, as y and z are obviously not equivalent in any nontrivial language.

As the above-given example shows, in order to be able to prove many simple equalities between open terms, it is necessary to develop techniques which allow us to deal with the target variables in ruloids in a reasonable way. In particular, we should not give too much importance to the names of target variables in ruloids.

Definition 2.3.8 (Valid ruloids) *Let G be a GSOS language and $P \in \mathbb{T}(\Sigma_G)$. We say that a ruloid $\rho = \dfrac{H}{P \xrightarrow{a} P'}$ is* valid *for P iff there exist $\rho' \in R_G(P)$ and an injective map $\sigma : \mathsf{TV}(\rho') \to (\mathsf{Var} - \mathsf{SV}(\rho))$ such that ρ is identical to $\rho'\sigma$.*

For example, it is immediate to notice that the rules

$$\frac{x \xrightarrow{a} z}{f(x) \xrightarrow{a} z} \qquad \frac{x \xrightarrow{a} y}{g(x) \xrightarrow{a} y}$$

are valid for the contexts $f(x)$ and $g(x)$ in the above-given example.

As a further example, consider the unary operation h with rule

$$\frac{x \xrightarrow{a} y_1,\ x \xrightarrow{a} y_2}{h(x) \xrightarrow{a} h(y_1)}$$

Applying the above definition, we immediately have that the rule

$$\frac{x \xrightarrow{a} y_1,\ x \xrightarrow{a} y_2}{h(x) \xrightarrow{a} h(y_2)}$$

is valid for f. (Just consider a substitution which swaps y_1 with y_2 in the rule for h.)

Note, moreover, that each ruloid in $R_G(P)$ is a valid ruloid for P.

The following lemma states that, if ρ' is obtained from ρ as in Definition 2.3.8, then ρ and ρ' are, in a sense, semantically equivalent ruloids.

Lemma 2.3.9 *Let $G = (\Sigma_G, R_G)$ be a GSOS language and $P \in \mathbb{T}(\Sigma_G)$. Assume that ρ is a valid ruloid for P because $\rho = \rho'\sigma$ for some $\rho' \in R_G(P)$ and injective $\sigma : \mathsf{TV}(\rho') \to \mathsf{Var} - \mathsf{SV}(\rho)$. Then:*

1. *ρ is sound for $\to_G$;*

2. $\mathsf{Supp}(\rho) = \mathsf{Supp}(\rho')$, *where, for a GSOS rule/ruloid* $\hat{\rho}$, $\mathsf{Supp}(\hat{\rho})$ *denotes the set of transitions supported by* $\hat{\rho}$.

The set of valid ruloids for a context P is infinite. However, by Theorem 2.3.2, we can always select a finite set of valid ruloids for P which is sound and supporting for it. We will often make use of this observation in what follows.

2.4 A logic of transition formulae

The set of ruloids associated with an open term P in a GSOS language characterizes its behaviour in much the same way as GSOS rules give the behaviour of GSOS operations. In fact, by Theorem 2.3.2, every transition from a closed term of the form $P\sigma$ can be inferred from a ruloid in $R_G(P)$.

The antecedents of ruloids give the precise conditions under which ruloids fire. When matching ruloids in the definition of the bisimulation-like relation between open terms that we aim at defining, we will let a ruloid ρ be matched by a set of ruloids J only if the antecedents of ρ are stronger than those of the ruloids in J, *i.e.*, if whenever ρ can fire under a substitution σ, then at least one of the ruloids in J can. In order to formalize this idea, we will make use of a simple propositional logic of initial transition formulae.

We define the language of *initial transition formulae* to be propositional logic with propositions of the form $x \xrightarrow{a}$. Formally, the formulae of such a logic are given by the following grammar:

$$F ::= \quad \mathsf{True} \mid x \xrightarrow{a} \mid \neg F \mid F \wedge F \ .$$

As usual, we write False for $\neg\mathsf{True}$, and $F \vee F'$ for $\neg(\neg F \wedge \neg F')$.

Let G be a GSOS language. A G-model for initial transition formulae is a substitution σ of processes (closed Σ_G-terms) for variables. We write $\rightarrow_G, \sigma \models F$ if the closed substitution σ is a model of the initial transition formula F. The satisfaction relation $\models$ is defined by structural recursion on F in the obvious way.

$$\begin{array}{rcl}
\rightarrow_G, \sigma \models \mathsf{True} & & \text{always} \\
\rightarrow_G, \sigma \models x \xrightarrow{a} & \Leftrightarrow & \sigma(x) \xrightarrow{a}_G p \text{ for some } p \\
\rightarrow_G, \sigma \models \neg F & \Leftrightarrow & \text{not } \rightarrow_G, \sigma \models F \\
\rightarrow_G, \sigma \models F \wedge F' & \Leftrightarrow & \rightarrow_G, \sigma \models F \text{ and } \rightarrow_G, \sigma \models F'
\end{array}$$

In what follows, we consider formulae up to commutativity and associativity of $\vee$ and $\wedge$, and we remove True conjuncts from formulae.

The reader familiar with Hennessy-Milner logic [74] will have noticed that the propositions of the form $x \xrightarrow{a}$ correspond to Hennessy-Milner formulae of the form $\langle a \rangle \mathsf{True}$.

If H is a finite set of positive or negative transition formulae (*e.g.*, the hypotheses of a rule or ruloid), then $\mathsf{hyps}(H)$ is the conjunction of the corresponding initial transition formulae. Formally,

$$\begin{array}{rcl} \mathsf{hyps}(\emptyset) & = & \mathsf{True} \\ \mathsf{hyps}(\left\{x \overset{a}{\nrightarrow}\right\} \cup H) & = & \neg(x \xrightarrow{a}) \wedge \mathsf{hyps}(H \setminus \left\{x \overset{a}{\nrightarrow}\right\}) \\ \mathsf{hyps}(\left\{x \xrightarrow{a} x'\right\} \cup H) & = & (x \xrightarrow{a}) \wedge \mathsf{hyps}(H \setminus \left\{x \xrightarrow{a} x'\right\}) \ . \end{array}$$

For example, $\mathsf{hyps}(\left\{x \xrightarrow{a} y, z \overset{b}{\nrightarrow}\right\}) = (x \xrightarrow{a}) \wedge \neg(z \xrightarrow{b})$. If J is a finite set of ruloids, we overload $\mathsf{hyps}(\cdot)$ and write:

$$\mathsf{hyps}(J) \stackrel{\Delta}{=} \bigvee_{\rho' \in J} \mathsf{hyps}(\mathsf{ante}(\rho')) \ . \tag{2.5}$$

The semantic entailment preorder between initial transition formulae may be now defined in the standard way; for formulae F, F', we have $\models_G F \Rightarrow F'$ iff every substitution that satisfies F must also satisfy F'. Note that a GSOS rule ρ is junk iff $\mathsf{hyps}(\rho)$ is semantically equivalent to False.

In the remainder, we will use the semantic entailment preorder between transition formulae in our test for equivalence of open terms to characterize the fact that if one ruloid may fire, then some other may do so too. Of course, in order to do so, we need to be able to check effectively when $\models_G F \Rightarrow F'$ holds. Fortunately, the semantic entailment preorder between formulae is decidable, as the following theorem shows.

Theorem 2.4.1 *Let G be a GSOS language. Then for all formulae F and F', it is decidable whether $\models_G F \Rightarrow F'$ holds.*

Proof. (**Sketch**) Let F and F' be formulae in the propositional language of initial transition formulae. For each mapping $\eta : \mathit{vars}(F) \cup \mathit{vars}(F') \to 2^{\mathsf{Act}}$ and $x \in \mathit{vars}(F) \cup \mathit{vars}(F')$, define

$$\eta \models'_G (x \xrightarrow{a}) \quad \Leftrightarrow \quad a \in \eta(x) \ .$$

We can extend $\models'_G$ to arbitrary formulae with variables in $vars(F) \cup vars(F')$ in the obvious way.

It is easy to see that, for all $\sigma : \mathsf{Var} \to \mathrm{T}(\Sigma_G)$ and formulae F'' over $vars(F) \cup vars(F')$,

$$\to_G, \sigma \models_G F'' \text{ iff } \eta_\sigma \models'_G F'',$$

where $\eta_\sigma(x) = init(\sigma(x))$ for all $x \in vars(F) \cup vars(F')$.

To see that our claim does hold, it is now sufficient to note that we can therefore reduce the refinement problem to checking that for every mapping $\eta : vars(F) \cup vars(F') \to init(\mathrm{T}(\Sigma_G))$, $\eta \models'_G F$ implies that $\eta \models'_G F'$. This problem is obviously decidable as there are only finitely many such mappings, and $init(\mathrm{T}(\Sigma_G))$ can be computed following the strategy presented in the proof of Theorem 2.2.9.

Theorem 2.4.1 tells us that we can safely use semantic entailment between formulae in our simple propositional language in the test for the validity of open equations in GSOS languages, which we will present in what follows.

2.5 Rule-matching bisimulation

We will now give a method to check the validity of equations in the algebra $\mathsf{Bisim}(G)$ based on a variation on the bisimulation technique. Our approach has strong similarities with, and is a sharpening of, *FH-bisimulation*, as proposed by de Simone in [51, 52]. (We remark, in passing, that FH-bisimilarity checking has been implemented in the tool ECRINS [53, 89].)

Definition 2.5.1 (Rule-matching bisimulation) *Let G be a GSOS language. A relation $\approx \subseteq \mathbb{T}(\Sigma_G) \times \mathbb{T}(\Sigma_G)$ is a* rule-matching bisimulation *if it is symmetric and $P \approx Q$ implies*

for each ruloid $\dfrac{H}{P \xrightarrow{a} P'}$ in the ruloid set of P, there exists a finite set J of valid ruloids for Q such that:

1. *For every $\rho' = \frac{H'}{Q \xrightarrow{a'} Q'} \in J$, we have:*
   (a) *$a' = a$.*
   (b) *$P' \approx Q'$.*
   (c) *$(TV(\rho') \cup TV(\rho)) \cap (SV(\rho) \cup SV(\rho')) = \emptyset$.*

(d) *If* $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$, *then* $x \xrightarrow{b} y \in H \cap H'$ *for some source variable* $x \in \mathsf{SV}(\rho) \cap \mathsf{SV}(\rho')$ *and action* b.

2. $\models_G \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(J)$.

We write $P \mathrel{\hat{\leftrightarrow}}_G Q$ *if there exists a rule-matching bisimulation* $\approx$ *relating* P *and* Q. *We sometimes refer to the relation* $\mathrel{\hat{\leftrightarrow}}_G$ *as* rule-matching bisimilarity.

Note that, as the source and target variables of GSOS rules and ruloids are distinct, condition 1c is equivalent to $\mathsf{TV}(\rho) \cap \mathsf{SV}(\rho') = \emptyset$ and $\mathsf{TV}(\rho') \cap \mathsf{SV}(\rho) = \emptyset$. Moreover, $\mathrel{\hat{\leftrightarrow}}_G$ is just standard bisimilarity over closed terms.

Remark 2.5.2 *Note that it is easy to handle junk ruloids when establishing a rule-matching bisimilarity. Indeed, if* ρ *is a junk ruloid then* $\mathsf{hyps}(\rho)$ *is semantically equivalent to* False. *Therefore, we can use an empty set* J *of ruloids to 'match* ρ*' and satisfy requirement 2 in Definition 2.5.1.*

Of course, the notion of rule-matching bisimulation is reasonable only if we can prove that it is sound with respect to the standard extension of bisimulation equivalence to open terms. This is the import of the following theorem, whose proof is in Section 2.10.

Theorem 2.5.3 (Soundness) *Let* G *be a GSOS language. Then, for all* $P, Q \in \mathbb{T}(\Sigma_G)$, $P \mathrel{\hat{\leftrightarrow}}_G Q$ *implies* $\mathsf{Bisim}(G) \models P = Q$.

The import of the above theorem is that, when trying to establish the equivalence of two contexts P and Q in a GSOS language G, it is sufficient to exhibit a rule-matching bisimulation relating them. A natural question to ask is whether the notion of rule-matching bisimulation is *complete* with respect to equality in $\mathsf{Bisim}(G)$, *i.e.* whether $\mathsf{Bisim}(G) \models P = Q$ implies $P \mathrel{\hat{\leftrightarrow}} Q$, for all $P, Q \in \mathbb{T}(\Sigma_G)$. Below, we shall provide a counter-example to the above statement.

Example 2.5.4 *Consider a GSOS language* G *consisting of a constant* $(a+b)^\omega$ *with rules*

$$\frac{}{(a+b)^\omega \xrightarrow{a} (a+b)^\omega} \qquad \frac{}{(a+b)^\omega \xrightarrow{b} (a+b)^\omega}$$

and unary function symbols f, g, h *and* i *with rules*

$$\frac{x \xrightarrow{a} y_1, x \xrightarrow{b} y_2}{h(x) \xrightarrow{a} f(x)} \qquad \frac{x \xrightarrow{a} y_1, x \xrightarrow{b} y_2}{i(x) \xrightarrow{a} g(x)} \qquad \frac{x \xrightarrow{a} y_1, x \xrightarrow{b} y_2}{f(x) \xrightarrow{a} f(x)} \qquad \frac{x \xrightarrow{a} y_1}{g(x) \xrightarrow{a} g(x)}$$

First of all, note that no rule in G is junk as the hypotheses of each of the above rules are satisfiable.

We claim that $\mathsf{Bisim}(G) \models h(x) = i(x)$. *To see this, it is sufficient to note that, for all* $p \in \mathrm{T}(\Sigma_G)$,

$$\begin{aligned} h(p) \xrightarrow{c} r \quad &\Leftrightarrow \quad p \xrightarrow{a}, \; p \xrightarrow{b}, \; c = a \text{ and } r \equiv f(p) \\ i(p) \xrightarrow{c} r \quad &\Leftrightarrow \quad p \xrightarrow{a}, \; p \xrightarrow{b}, \; c = a \text{ and } r \equiv g(p) \end{aligned}$$

Moreover, for a term p such that $a, b \in \mathrm{init}(p)$, *it is immediate to see that* $f(p) \leftrightarrows g(p)$ *as both these terms can only perform action a indefinitely.*

However, $h(x)$ *and* $i(x)$ *are* not *rule-matching bisimilar. In fact, in order for* $h(x) \leftrightarrows i(x)$ *to hold, it must be the case that* $f(x) \leftrightarrows g(x)$. *This does not hold as the unique rule for* $g(x)$ *cannot be matched by the rule for* $f(x)$ *because* $\not\models (x \xrightarrow{a}) \Rightarrow (x \xrightarrow{a} \wedge\, x \xrightarrow{b})$. *Take,* e.g., *a closed substitution* σ *such that* $\sigma(x) \equiv h((a + b)^\omega)$.

Intuitively, the failure of rule-matching bisimulation in the above example is due to the fact that, in order for $\mathsf{Bisim}(G) \models h(x) = i(x)$ *to hold, it is sufficient that* $f(p)$ *and* $g(p)$ *be bisimilar for those terms p which enable transitions from* $h(p)$ *and* $i(p)$, *rather than for arbitrary instantiations.*

The above example used GSOS rules which contain multiple positive antecedents for the same variable. We will now provide a counter-example to the completeness of rule-matching bisimulation which uses only the format of rules due to de Simone [51, 52]. The point of the example is to show that, in general, one needs some kind of *semantic* information in establishing the equivalence of two contexts.

Example 2.5.5 *Consider a GSOS language G consisting of the constants* $a, b, \mathbf{0}$ *with rules*

$$\frac{}{a \xrightarrow{a} \mathbf{0}} \qquad \frac{}{b \xrightarrow{b} \mathbf{0}}$$

and unary function symbols f, g, f' *and* g' *with rules*

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f'(y)} \qquad \frac{x \xrightarrow{a} y}{g(x) \xrightarrow{a} g'(y)}$$

$$\frac{x \xrightarrow{b} y}{f'(x) \xrightarrow{b} \mathbf{0}} \qquad \frac{}{f'(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{}{g'(x) \xrightarrow{a} \mathbf{0}}$$

First of all, note that no rule in G is junk as the hypotheses of each of the above rules are satisfiable.

We claim that $\mathsf{Bisim}(G) \models f(x) = g(x)$. *To see this, it is sufficient to note that, for all* $p \in \mathrm{T}(\Sigma_G)$,

1. $f(p) \xrightarrow{a} f'(p') \Leftrightarrow p \xrightarrow{a} p' \Leftrightarrow g(p) \xrightarrow{a} g'(p')$,
2. $p \xrightarrow{a} p'$ *implies* $p' \overset{b}{\not\to}$, *and*
3. $p \overset{b}{\not\to}$ *implies* $f'(p) \leftrightarrows a \leftrightarrows g'(p)$.

However, $f(x)$ *and* $g(x)$ *are* not *rule-matching bisimilar. In fact, in order for* $f(x) \mathrel{\reflectbox{$\leftrightarrows$}} g(x)$ *to hold, it must be the case that* $f'(y) \mathrel{\reflectbox{$\leftrightarrows$}} g'(y)$. *This does not hold as the rule*

$$\frac{y \xrightarrow{b} y'}{f'(y) \xrightarrow{b} \mathbf{0}}$$

for $f'(y)$ *cannot be matched by the single axiom for* $g'(y)$, *as they have a different action.*

As our readers can easily check, the above example gives an instance of an equation that holds in a language G, but not in all of its disjoint extensions. On the other hand, note that the equation discussed in Example 2.5.4 is valid in each disjoint extension of the GSOS language considered there.

In the following section we will provide examples that will, hopefully, convince our readers that rule-matching bisimulation is a tool which, albeit not complete, can be used to check the validity of many interesting equations.

It is natural to ask oneself at this point whether rule-matching bisimilarity is preserved by taking disjoint extensions, *i.e.*, whether an equation that has been proven to hold in a language G using rule-matching bisimilarity remains sound for each disjoint extension of G. The following example shows that this is not the case.

Example 2.5.6 *Consider a GSOS language G consisting of a constant a^ω with rule $a^\omega \xrightarrow{a} a^\omega$ and unary operations f and g with the following rules.*

$$\frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} f(x)} \qquad \frac{y \xrightarrow{a} y'}{g(y) \xrightarrow{a} g(y)}$$

First of all, note that no rule in G is junk as the hypotheses of each of the above rules are satisfiable.

We claim that $\mathsf{Bisim}(G) \models f(x) = g(y)$. *To see this, it is sufficient to note that each closed term in the language is bisimilar to* a^ω. *Moreover,* $f(x) \leftrightarrows_G g(y)$ *holds because the formulae* $x \xrightarrow{a}$ *and* $y \xrightarrow{a}$ *are logically equivalent in* G. *On the other hand, consider the disjoint extension* G' *of* G *obtained by adding the constant* $\mathbf{0}$ *with no rules to* G. *In this disjoint extension,* $f(x) \not\leftrightarrows_{G'} g(y)$ *does* not *hold because* $x \xrightarrow{a}$ *does not entail* $y \xrightarrow{a}$.

However, rule-matching bisimilarity in language G *is* preserved by taking disjoint extensions if the language G is sufficiently expressive in the sense formalized by the following result. (Recall that $\mathsf{BISIM}(G)$ denotes the class of all algebras $\mathsf{Bisim}(G')$, for G' a disjoint extension of G.)

Theorem 2.5.7 *Let G be a GSOS language such that* $\mathrm{init}(\mathrm{T}(\Sigma_G)) = 2^{\mathsf{Act}}$. *Then, for all* $P, Q \in \mathbb{T}(\Sigma_G)$, $P \leftrightarrows_G Q$ *implies* $\mathsf{BISIM}(G) \models P = Q$.

Proof. The proof of Theorem 2.5.3 can be replayed, making use of the observation that, for each disjoint extension G' of G, the collection of ruloids in G' for a Σ_G-term P coincides with the collection of ruloids for P in G. Moreover, in light of the proviso of the theorem, $\models_G F \Rightarrow F'$ iff $\models_{G'} F \Rightarrow F'$, for all formulae F and F'.

A conceptually interesting consequence of the above result is that, when applied to a sufficiently expressive GSOS language G, rule-matching bisimilarity is a proof method that is, in some sense, *monotonic with respect to taking disjoint extensions of the original language*. This means that rule-matching bisimilarity can only prove the validity of equations in G that remain true in all its disjoint extensions. A similar limitation applies to the proof methods presented in, *e.g.*, [52, 149].

The condition on the set $\mathit{init}(\mathrm{T}(\Sigma_G))$ in the statement of Theorem 2.5.7 above ensures that each semantic entailment $F \Rightarrow F'$ that holds in the GSOS language G holds also in all its disjoint extensions. Let us say that an entailment with the above property is *G-robust*. A rule-matching bisimulation over G is G-robust if so are all the entailments that need to be checked in item 2 in Definition 2.5.1. We can now formulate the following sharpening of Theorem 2.5.7.

Theorem 2.5.8 *Let G be a GSOS language.*

1. *Assume that $\approx$ is a G-robust rule-matching bisimulation over G, and let G′ be a disjoint extension of G. Then $\approx$ is a rule-matching bisimulation over G′.*
2. *Let $P, Q \in \mathbb{T}(\Sigma_G)$. Assume that $P \mathrel{\hat{\leftrightarrow}}_G Q$ can be shown by establishing a G-robust rule-matching bisimulation over G. Then $\mathsf{BISIM}(G) \models P = Q$.*

Proof. The former claim can be shown mimicking the proof of Theorem 2.5.7. The latter follows immediately from the former and Theorem 2.5.3.

As our reader will notice, all the examples of rule-matching bisimulations we consider in the following section are robust. Indeed, their robustness can be checked syntactically by using just the well known validity of the entailment $F \wedge F' \Rightarrow F$. Therefore, in light of Theorem 2.5.8, the equivalences we discuss hold in all the disjoint extensions of the considered language fragments.

2.6 Examples

We shall now present some examples of applications of the 'rule-matching bisimulation technique'. In particular, we shall show how some well known equations found in the literature on process algebra can be verified using it.

Commutativity of choice in BCCSP Let BCCSP [146, 96] be the GSOS language containing the constant $\mathbf{0}$, unary prefixing operators $a.\_$ ($a \in \mathsf{Act}$) and the binary choice operator + with the following rules.

$$\frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \tag{2.6}$$

It is well known that the equality $x + y = y + x$ holds in each disjoint extension of BCCSP. This can be easily shown using rule-matching bisimilarity. Indeed, the ruloids for the contexts $x + y$ and $y + x$ are

$$\frac{x \xrightarrow{a} x'}{\begin{array}{l} x + y \xrightarrow{a} x' \\ y + x \xrightarrow{a} x' \end{array}} \qquad \frac{y \xrightarrow{a} y'}{\begin{array}{l} x + y \xrightarrow{a} y' \\ y + x \xrightarrow{a} y' \end{array}}$$

Therefore, as our reader can easily check, the relation

$$\approx \;\stackrel{\Delta}{=}\; \{(x+y, y+x), (y+x, x+y)\} \cup \{(z,z) \mid z \in \mathsf{Var}\}$$

is a rule-matching bisimulation.

Associativity of sequencing Let G be any GSOS language containing the sequencing operation specified by rules (2.3) on page 29. Let $R[x,y,z] = x;(y;z)$ and $L[x,y,z] = (x;y);z$. The ruloids for these two contexts were given by rules by rules (2.3.5) on page 30.

Consider the symmetric closure of the relation

$$\approx \;\stackrel{\Delta}{=}\; \{(R[x,y,z], L[x,y,z]) \mid x,y,z \in \mathsf{Var}\} \cup \mathcal{I}$$

where $\mathcal{I}$ denotes the identity relation over $\mathbb{T}(\Sigma_G)$. By Theorem 2.5.3, to show that the contexts L and R are equivalent, it is sufficient to check that what we have just defined is a rule-matching bisimulation. In particular, we need to check the correspondence between the ruloids for these contexts (which is the one given in (2.3.5)), and then check that the targets are related by $\approx$. The verification of these facts is trivial. Thus we have shown that sequencing is associative in any GSOS language that contains the sequencing operation.

The associativity proofs for the standard parallel composition operators found in *e.g.* ACP, CCS, SCCS and Meije, and for the choice operators in those calculi, for example the associativity of '+' in BCCSP, follow similar lines.

Commutativity of interleaving parallel composition Many standard axiomatizations of behavioural equivalences in the literature, such as the ones offered in [74], cannot be used to show by purely equational means that, *e.g.*, parallel composition is commutative and associative. We will now show how this can be easily done using the rule-matching bisimulation technique. We will exemplify the methods by showing that the interleaving parallel composition operation $\|$ [77] is commutative.

We recall that the rules for $\|$ are (one pair of rules for each $a \in \mathsf{Act}$):

$$\frac{x \stackrel{a}{\to} x'}{x \parallel y \stackrel{a}{\to} x' \parallel y} \qquad \frac{y \stackrel{a}{\to} y'}{x \parallel y \stackrel{a}{\to} x \parallel y'} \tag{2.7}$$

The ruloids for the contexts $x \lll y$ and $y \lll x$ given by Theorem 2.3.2 are (one pair of ruloids for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x'}{\begin{array}{c} x \lll y \xrightarrow{a} x' \lll y \\ y \lll x \xrightarrow{a} y \lll x' \end{array}} \qquad \frac{y \xrightarrow{a} y'}{\begin{array}{c} x \lll y \xrightarrow{a} x \lll y' \\ y \lll x \xrightarrow{a} y' \lll x \end{array}}$$

It is now immediate to see that the relation $\{(x \lll y, y \lll x) \mid x, y \in \mathsf{Var}\}$ is a rule-matching bisimulation in any GSOS language that includes the interleaving operator. In fact, the correspondence between the ruloids is trivial and the targets are related by the above relation.

A distributivity law for sequencing We can use the 'rule-matching bisimulation' method to verify the validity of a distributivity axiom for sequencing which has been presented in [30].

Let G be a GSOS language which extends $\mathsf{FINTREE}$ with the sequencing operation defined by the rules in Example 2.3.5. The law whose validity we want to check is the following

$$(a.x + b.y + y');z \quad = \quad a.(x;z) + (b.y + y');z \tag{2.8}$$

Let $L[x, y, y', z] \equiv (a.x + b.y + y');z$ and $R[x, y, y', z] \equiv a.(x;z) + (b.y + y');z$. First of all, let us list the ruloids for the contexts $L[x, y, y', z]$ and $R[x, y, y', z]$. The ruloids for L and R are

$$\frac{}{\begin{array}{c} L \xrightarrow{a} x;z \\ R \xrightarrow{a} x;z \end{array}} \qquad \frac{}{\begin{array}{c} L \xrightarrow{b} y;z \\ R \xrightarrow{b} y;z \end{array}} \qquad \frac{y' \xrightarrow{c} y''}{\begin{array}{c} L \xrightarrow{c} y'';z \\ R \xrightarrow{c} y'';z \end{array}}$$

where c can be any action in Act. Matching the ruloids for the contexts L and R as explicitly given above, it is easy to show that the relation $\{(L, R), (R, L)\} \cup \mathcal{I}$ is a rule-matching bisimulation in any GSOS language that includes the above-mentioned operators.

De Simone's clock example In his seminal paper [52], de Simone presents a bisimulation based technique useful for proving open equations between contexts specified using the so-called de Simone format of operational rules. On page 260 of

that paper, de Simone discusses two examples showing that there are valid open equalities between contexts that his technique cannot handle. Below, we shall discuss a variation on one of his examples, the *clock example*, which maintains all the characteristics of the original one in [52], showing how rule-matching bisimulations can be used to check the relevant equalities.

In order to proceed with the example, we show the rules describing the SOS semantics of the parallel composition operator with synchronization.

Fix a partial, commutative and associative function $\gamma : \mathsf{Act} \times \mathsf{Act} \rightharpoonup \mathsf{Act}$, which describes the synchronization between actions. The $\|$ operation can be described by the rules (for all $a, b, c \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y} \qquad \frac{y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x\|y'} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y'}{x\|y \xrightarrow{c} x'\|y'}\ \gamma(a,b) = c$$

Suppose now that we have a GSOS language G which includes parallel composition with synchronization, $\|$, the interleaving operation, $\|\!|$, described by the rules (2.7), and a constant Ω_{Act} (the *clock over the whole set of actions* in de Simone's terminology) with rules

$$\Omega_{\mathsf{Act}} \xrightarrow{a} \Omega_{\mathsf{Act}} \quad (a \in \mathsf{Act})\ .$$

Consider the contexts $C[x] \equiv x\|\Omega_{\mathsf{Act}}$ and $D[x] \equiv x \,\|\!|\, \Omega_{\mathsf{Act}}$. We do have that, regardless of the precise description of G, the terms $C[x]$, $D[x]$ and Ω_{Act} are all equal in $\mathsf{Bisim}(G)$. This can be easily shown by establishing that the symmetric closures of the relations $\{(C[p], \Omega_{\mathsf{Act}}) \mid p \in \mathrm{T}(\Sigma_G)\}$ and $\{(D[p], \Omega_{\mathsf{Act}}) \mid p \in \mathrm{T}(\Sigma_G)\}$ are bisimulations. However, as argued in [52], de Simone's techniques based on FH-bisimilarity cannot be used to establish these equalities. We can instead establish their validity using our rule-matching bisimulation technique as follows.

First of all, we compute the ruloids for the contexts $C[x]$ and $D[x]$. These are, respectively,

$$\frac{}{C[x] \xrightarrow{a} C[x]}\ (a \in \mathsf{Act}) \qquad \frac{x \xrightarrow{a} x'}{C[x] \xrightarrow{a} C[x']}\ (a \in \mathsf{Act}) \qquad \frac{x \xrightarrow{a} x'}{C[x] \xrightarrow{b} C[x']}\ \exists c \in \mathsf{Act} : \gamma(a,c) = b$$

and

$$\frac{}{D[x] \xrightarrow{a} D[x]}\ (a \in \mathsf{Act}) \qquad \frac{x \xrightarrow{a} x'}{D[x] \xrightarrow{a} D[x']}\ (a \in \mathsf{Act})\ .$$

Now, it can be easily checked that the symmetric closure of the relation

$$\{(C[x], D[z]), (C[x], \Omega_{\mathsf{Act}}), (D[x], \Omega_{\mathsf{Act}}) \mid x, z \in \mathsf{Var}\}$$

is a rule-matching bisimulation. The point is that *any* ruloid for $C[x]$ can be matched by an axiom for $D[z]$, and, vice versa, *any* ruloid for $D[z]$ can be matched by an axiom for $C[x]$. This is because it is always the case that $\models (x \xrightarrow{a}) \Rightarrow \mathsf{True}$ for $x \in \mathsf{Var}$.

Some equations for while loops One of the features of GSOS rules is the fact that they allow for copying of arguments, *e.g.*, arguments that are tested positively in a GSOS rule may appear in the target of a rule. We recall that copying of arguments is not allowed in the format due to de Simone [51, 52]. An operation whose rules use such a feature is a simple kind of while-loop for concurrent processes.

Pick two distinguished action t and f in Act. Then we can define a binary operation $\mathsf{while}(\_,\_)$ which runs its second argument if the first can emit a t action. Formally, the rules for $\mathsf{while}(\_,\_)$ are (one such rule for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{t} x',\ y \xrightarrow{a} y'}{\mathsf{while}(x, y) \xrightarrow{a} y'; \mathsf{while}(x', y)}$$

where ';' is the sequencing operation described previously.

Let us now define a few other operations. An unconditional looping construct can be given by the following rules (one such rule for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x'}{\mathsf{loop}(x) \xrightarrow{a} x'; \mathsf{loop}(x)}$$

Note that also the rules for the $\mathsf{loop}(\_)$ operation use copying of arguments.

Let t^ω denote a constant with behaviour given by the rule:

$$t^\omega \xrightarrow{t} t^\omega$$

Let G be any GSOS language which disjointly extends BCCSP with these operations. Then, using the rule-matching bisimulation technique, we can easily prove

that the following identities hold in $\mathsf{Bisim}(G)$.

$$\begin{aligned} \mathsf{while}(t^\omega, y) &= \mathsf{loop}(y) \\ \mathsf{while}(f.x, y) &= \mathbf{0} \\ \mathsf{while}(t.x, y) &= y; \mathsf{while}(x, y) \end{aligned}$$

By way of example, consider the equation

$$\mathsf{while}(t^\omega, y) = \mathsf{loop}(y) \ .$$

The ruloids for the contexts arising from $\mathsf{while}(t^\omega, y)$ and $\mathsf{loop}(y)$ are listed below. (There is one ruloid for each $a \in \mathsf{Act}$.)

$$\frac{y \xrightarrow{a} y'}{\mathsf{while}(t^\omega, y) \xrightarrow{a} y'; \mathsf{while}(t^\omega, y)} \qquad \frac{y' \xrightarrow{a} y''}{y'; \mathsf{while}(t^\omega, y) \xrightarrow{a} y''; \mathsf{while}(t^\omega, y)}$$

$$\frac{y' \xrightarrow{t}, \ y \xrightarrow{a} y''}{y'; \mathsf{while}(t^\omega, y) \xrightarrow{a} y''; \mathsf{while}(t^\omega, y)}$$

$$\frac{y \xrightarrow{a} y'}{\mathsf{loop}(y) \xrightarrow{a} y'; \mathsf{loop}(y)} \qquad \frac{y' \xrightarrow{a} y''}{y'; \mathsf{loop}(y) \xrightarrow{a} y''; \mathsf{loop}(y)} \qquad \frac{y' \xrightarrow{t}, \ y \xrightarrow{a} y''}{y'; \mathsf{loop}(y) \xrightarrow{a} y''; \mathsf{loop}(y)}$$

As our reader can easily check, the symmetric closure of the relation

$$\{(\mathsf{while}(t^\omega, y), \mathsf{loop}(y))\} \cup \{(z; \mathsf{while}(t^\omega, y), z; \mathsf{loop}(y)) \mid z \in \mathsf{Var}\}$$

is a rule-matching bisimulation. Indeed, the above-listed ruloid types with identical premises match one by one.

2.7 Partial completeness results

In previous sections, we showed that the rule-matching bisimulation technique, albeit not complete in general, can be used to prove several important equations found in the literature on process algebras. In particular, the soundness of all the equations generated by the methods in [5] can be proven by exhibiting appropriate rule-matching bisimulations. A natural question to ask is whether there are some

classes of contexts for which rule-matching bisimulations give us a complete proof technique for establishing equality between contexts. One such class of contexts is, of course, that of closed terms, as rule-matching bisimilarity coincides with bisimilarity over processes.

Below we will present another partial completeness result, this time with respect to a class of contexts that we call 'persistent'.

Definition 2.7.1 *Let G be a GSOS language and $P \in \mathbb{T}(\Sigma_G)$. We say that P is* persistent *iff each ruloid in $R_G(P)$ is of the form $\frac{H}{P \xrightarrow{a} P}$ for some $a \in \mathsf{Act}$.*

Thus persistent contexts are terms that test their arguments, perform actions according to the results of these tests, and then remain unchanged.

Theorem 2.7.2 (Completeness for persistent contexts) *Let G be a GSOS language. Then $\mathsf{Bisim}(G) \models P = Q$ iff $P \leftrightarrows Q$, for all persistent $P, Q \in \mathbb{T}(\Sigma_G)$.*

The proof of the above result may be found in Section 2.11.

We now proceed to introduce another class of operations for which rule-matching bisimilarity yields a complete proof method.

Definition 2.7.3 (Non-inheriting rule) *A GSOS rule of the form (2.1) on page 22 is* non-inheriting *if none of the variables in $\vec{x}$, namely the source variables in the rule, occurs in the target of the conclusion of the rule $C[\vec{x}, \vec{y}]$. A GSOS language is* non-inheriting *if so is each of its rules. Non-inheriting de Simone rules and languages are defined similarly.*

In particular a non-inheriting rule in de Simone format has the following form:

$$\frac{\left\{x_i \xrightarrow{a_i} y_i \mid i \in I\right\}}{f(x_1, \ldots, x_n) \xrightarrow{c} t} \ ,$$

where $I \subseteq \{1, \ldots, n\}$, all the variables x_i and y_i are distinct, a_i and c are actions, f is an operation symbol from Σ with arity n, and t is a term such that $vars(t) \subseteq \{y_i \mid i \in I\}$ and each variable occurs in t at most once.

Theorem 2.7.4 *Let G be a non-inheriting GSOS language that, for each $P \in \mathbb{T}(\Sigma_G)$ and $c \in \mathsf{Act}$, contains at most one ruloid for P having $c \in \mathsf{Act}$ as action. Let G' be the disjoint extension of G obtained by adding to G the operations and rules of the language BCCSP with Act as set of actions. Let P and Q be terms over Σ_G. Then $\mathsf{Bisim}(G') \models P = Q$ implies $P \leftrightarrows_{G'} Q$.*

A proof of the above theorem may be found in Section 2.12. A minor modification of that argument yields a partial completeness result for a class of de Simone systems.

Theorem 2.7.5 *Let G be a non-inheriting de Simone language that, for each $f \in \Sigma_G$ and $c \in \mathsf{Act}$, contains at most one rule having $f \in \Sigma_G$ as principal operation and $c \in \mathsf{Act}$ as action. Let G' be the disjoint extension of G obtained by adding to G the constant $\mathbf{0}$ and the Act-labelled prefixing operations from the language BCCSP. Let P and Q be terms over Σ_G. Then $\mathsf{Bisim}(G') \models P = Q$ implies $P \leftrightarrows_{G'} Q$.*

For instance, the above theorem yields that rule-matching bisimilarity can prove all the sound equations between terms constructed using variables and the operations of restriction and injective relabelling from CCS [96] and synchronous parallel composition from CSP [77].

2.8 Extending rule-matching bisimilarity to GSOS with predicates

In this section, we extend our main results to the setting of GSOS language specifications with predicates. This extension, albeit not particularly deep theoretically, is significant from the point of view of applications of rule-matching bisimilarity since the operational semantics of several operations commonly found in the literature on, *e.g.*, process algebra is best specified using rules involving the use of predicates as first-class notions.

2.8.1 GSOS with predicates

Given a signature Σ and a set $\mathcal{P}$ of predicate symbols, $Pr\ t$ is a *positive predicate formula* and $\neg Pr\ t$ is a *negative predicate formula*, for each $Pr \in \mathcal{P}$ and $t \in \mathbb{T}(\Sigma)$. The shape of the deduction rules in Definition 2.2.1 is adapted in order to prove also predicate formulae and to involve them in premises of rules.

Definition 2.8.1 (GSOS rule with predicates) *Suppose Σ is a signature and $\mathcal{P}$ is a set of predicate symbols. A* GSOS rule with predicates *over Σ and $\mathcal{P}$ is a rule of the form:*

$$\frac{\bigcup_{i=1}^{l}\left\{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq j \leq m_i\right\} \;\cup\; \bigcup_{i=1}^{l}\left\{x_i \xrightarrow{b_{ik}}\!\!\!\!\!\rightarrow \mid 1 \leq k \leq n_i\right\} \;\cup\; PH}{(f(x_1,\ldots,x_l) \xrightarrow{c} C[\vec{x},\vec{y}] \;\; or \;\; Pr\, f(x_1,\ldots,x_l))} \tag{2.9}$$

where $PH = \bigcup_{i=1}^{l}\{Pr_{ik}\, x_i \;\; or \;\; \neg Pr_{ik}\, x_i \mid 1 \leq k \leq o_i\}$, *all the variables are distinct,* $m_i, n_i, o_i \geq 0$, a_{ij}, b_{ik}, *and* c *are actions,* f *is an operation symbol from* Σ *with arity* l, Pr *and* Pr_{ik} *are predicate symbols and* $C[\vec{x},\vec{y}]$ *is a* Σ*-context.*

Definition 2.8.2 *A* GSOS language with predicates *is a triple* $G = (\Sigma_G, \mathcal{P}_G, R_G)$, *where* Σ_G *is a finite signature,* $\mathcal{P}_G$ *is a finite set of predicate symbols and* R_G *is a finite set of GSOS rules over* Σ_G *and* $\mathcal{P}_G$.

The transition relation $\rightarrow_G$ and the set PF_G of provable predicate formulae of the form $Pr\ p$, where p is a closed term, are the ones defined by the rules using structural induction over closed Σ_G-terms, as described in Section 2.2.

The definition of bisimulation is extended to a setting with predicates in the standard fashion. In particular, bisimilar terms must satisfy the same predicates.

Example 2.8.3 *As a classic example of an operator whose operational specification involves predicates, consider the standard formulation of the sequential composition operator '*$\cdot$*'. The rules below make use of the predicate symbol* $\downarrow$*. (Intuitively, the formula* $x \downarrow$ *means that* x *successfully terminates.)*

$$(seq1)\ \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad (seq2)\ \frac{x \downarrow \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'} \qquad (seq3)\ \frac{x \downarrow \quad y \downarrow}{(x \cdot y) \downarrow}$$

Remark 2.8.4 *The reader familiar with [32] may have noticed that the definition of '*$\cdot$*' fits the tagh format of Baeten and de Vink. In this format, GSOS languages are extended to involve a single predicate,* $\downarrow$*, which encodes an explicit notion of successful termination of terms. GSOS languages with predicates generalize this format by considering a finite set of predicates. In contrast to the tagh format, rules may also contain negative predicate formulae in premises and the rules with predicate formulae as conclusions may have transition formulae in premises. Moreover, since the signature may have more than one predicate symbol, in every rule a single argument can be tested more times in the context of different predicates.*

2.8.2 A ruloid theorem for GSOS languages with predicates

A *ruloid* for a context $D[\vec{x}]$, with $\vec{x} = (x_1, \ldots, x_l)$, takes the form:

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq j \leq m_i \right\} \;\cup\; \bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{b_{ik}} \mid 1 \leq k \leq n_i \right\} \;\cup\; PH}{(D[\vec{x}] \xrightarrow{c} C[\vec{x}, \vec{y}] \;\; or \;\; Pr\, D[\vec{x}])} \tag{2.10}$$

where $PH = \bigcup_{i=1}^{l} \{Pr_{ik}\, x_i \text{ or } \neg Pr_{ik}\, x_i \mid 1 \leq k \leq o_i\}$, all the variables are distinct, $m_i, n_i, o_i \geq 0$, a_{ij}, b_{ik}, and c are actions, Pr and Pr_{ik} are predicate symbols and $C[\vec{x}, \vec{y}]$ and $D[\vec{x}]$ are Σ-contexts.

The notion of supporting set of ruloids for a context $D[\vec{x}]$ and action c, introduced in Definition 2.3.1, is adapted to ruloids of the aforementioned form in the obvious way. Moreover, in a setting with predicates, we need to consider ruloids that are supporting for a context $D[\vec{x}]$ and a predicate symbol Pr.

Definition 2.8.5 *A set of ruloids R is* supporting *for a context $D[\vec{x}]$ and a predicate symbol Pr iff all the consequents of ruloids in R are of the form $Pr\, D[\vec{x}]$ and, whenever $Pr\, D[\vec{P}]$ holds, there are a ruloid $\rho \in R$ and a closed substitution σ such that $\mathsf{cons}(\rho)\sigma = Pr\, D[\vec{P}]$ and $\sigma \models \mathsf{ante}(\rho)$.*

The following result can be shown by mimicking the proof of the Ruloid Theorem (Theorem 7.4.3) in [44].

Theorem 2.8.6 (Ruloid theorem) *Let G be a GSOS language with predicates. For each $D[\vec{x}] \in \mathbb{T}(\Sigma_G)$ and action c, there exists a finite set $R_{D,c}$ of ruloids of the form (2.10) that is sound and supporting for $D[\vec{x}]$ and action c. Similarly, for each $D[\vec{x}] \in \mathbb{T}(\Sigma_G)$ and predicate symbol Pr, there exists a finite set $R_{D,Pr}$ of ruloids of the form (2.10) that is sound and supporting for $D[\vec{x}]$ and Pr.*

The notion of valid ruloid from Definition 2.3.8 and related results can be adapted naturally to a setting with predicates.

2.8.3 The logic of initial transitions with predicates

In this section we extend the logic of initial transition with predicates. Formulae are given by adding propositions of the form $Pr\, x$ to the grammar from Section 2.4.

$$F ::= \ldots \mid Pr\, x \;\; .$$

The semantics of formulae is extended to consider the new kind of predicate formulae and is presented below. We write $(\rightarrow_G \ \cup \ PF_G), \sigma \models F$ if the closed substitution σ is a model of the initial transition formula F, where PF_G denotes the set of provable predicate formulae in G.

$$
\begin{array}{rcl}
(\rightarrow_G \ \cup \ PF_G), \sigma \models \mathsf{True} & & \text{always} \\
(\rightarrow_G \ \cup \ PF_G), \sigma \models Pr\, x & \Leftrightarrow & Pr\, \sigma(x) \in PF_G \\
(\rightarrow_G \ \cup \ PF_G), \sigma \models x \xrightarrow{a} & \Leftrightarrow & \sigma(x) \xrightarrow{a}_G \\
(\rightarrow_G \ \cup \ PF_G), \sigma \models \neg F & \Leftrightarrow & \text{not } (\rightarrow_G \ \cup \ PF_G), \sigma \models F \\
(\rightarrow_G \ \cup \ PF_G), \sigma \models F \wedge F' & \Leftrightarrow & (\rightarrow_G \ \cup \ PF_G), \sigma \models F \text{ and } (\rightarrow_G \ \cup \ PF_G), \sigma \models F'
\end{array}
$$

If H is a set of transition or predicate formulae (*e.g.*, the hypotheses of a rule or ruloid with predicates), then $\mathsf{hyps}(H)$ is the conjunction of the corresponding initial transition formulae. Formally, the definition on page 33 is extended with the clauses

$$
\begin{array}{l}
\mathsf{hyps}(\{Pr\, x\} \cup H) = (Pr\, x) \wedge \mathsf{hyps}(H \setminus \{Pr\, x\}) \quad \text{and} \\
\mathsf{hyps}(\{\neg Pr\, x\} \cup H) = \neg(Pr\, x) \wedge \mathsf{hyps}(H \setminus \{\neg Pr\, x\}) \ .
\end{array}
$$

The notion of entailment between formulae in the logic with predicate formulae is defined as before.

Theorem 2.8.7 *Let G be a GSOS language with predicates. Then, for all formulae F and F', it is decidable whether $\models_G F \Rightarrow F'$ holds.*

Proof. (**Sketch**) Let F and F' be formulae in the propositional language of initial transition formulae with predicates. For each mapping $\eta : vars(F) \cup vars(F') \rightarrow 2^{\mathsf{Act} \cup \mathcal{P}}$ and $x \in vars(F) \cup vars(F')$, define

$$
\begin{array}{rcl}
\eta \models'_G (x \xrightarrow{a}) & \Leftrightarrow & a \in \eta(x) \\
\eta \models'_G Pr\, x & \Leftrightarrow & Pr \in \eta(x) \ .
\end{array}
$$

We extend $\models'_G$ to arbitrary formulae with variables in $vars(F) \cup vars(F')$ in the obvious way.

Now consider the set $init{+}pred(p)$, with $p \in \mathrm{T}(\Sigma_G)$, containing the labels of the initial transitions of the closed term p together with the names of the predicate

symbols that p satisfies. Formally, for each closed term p, we define $\mathit{init}{+}\mathit{pred}(p) = \mathit{init}(p) \cup \{Pr \in \mathcal{P} \mid Pr\ p\}$.

It is easy to see that, for all $\sigma : \mathsf{Var} \to \mathrm{T}(\Sigma_G)$ and formulae F'' over $\mathit{vars}(F) \cup \mathit{vars}(F')$,

$$(\to_G \ \cup\ PF_G), \sigma \models_G F'' \text{ iff } \eta_\sigma \models'_G F'',$$

where $\eta_\sigma(x) = \mathit{init}{+}\mathit{pred}(\sigma(x))$ for all $x \in \mathit{vars}(F) \cup \mathit{vars}(F')$.

To see that our claim does hold, it is now sufficient to note that we can therefore reduce the refinement problem to checking that, for every mapping $\eta : \mathit{vars}(F) \cup \mathit{vars}(F') \to \mathit{init}{+}\mathit{pred}(\mathrm{T}(\Sigma_G))$, if $\eta \models'_G F$ then $\eta \models'_G F'$. This problem is obviously decidable as there are only finitely many such mappings, and $\mathit{init}{+}\mathit{pred}(\mathrm{T}(\Sigma_G))$ can be effectively computed using an adaptation of the strategy presented in the proof of Theorem 2.2.9.

2.8.4 Rule-matching bisimilarity

In this section we reformulate the notion of rule-matching bisimilarity for GSOS languages with predicates. In the following definition, the reader may find it helpful to bear in mind that, in this setting, differently from GSOS languages, the ruloids for a context P may be of two types:

- ruloids of the form $\dfrac{H}{P \xrightarrow{a} P'}$ and
- ruloids of the form $\dfrac{H}{Pr\ P}$.

In GSOS languages ruloids take only the first form. Another important difference is that, in GSOS languages with predicates, the set of premises H can involve predicates.

Definition 2.8.8 (Rule-matching bisimulation, reprise) *Let G be a GSOS language with predicates. A relation $\approx\ \subseteq \mathbb{T}(\Sigma_G) \times \mathbb{T}(\Sigma_G)$ is a* rule-matching bisimulation *if it is symmetric and $P \approx Q$ implies that, for each ruloid ρ in the ruloid set of P, the following conditions are met.*

1. *If ρ is of the form $\dfrac{H}{P \xrightarrow{a} P'}$ then there exists a finite set J of valid ruloids for Q such that:*

- *For every* $\rho' = \frac{H'}{Q \xrightarrow{a'} Q'} \in J$, *we have:*
  - $a' = a$,
  - $P' \approx Q'$,
  - $(\mathsf{TV}(\rho') \cup \mathsf{TV}(\rho)) \cap (\mathsf{SV}(\rho) \cup \mathsf{SV}(\rho')) = \emptyset$ *and*
  - *if* $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$, *then* $x \xrightarrow{b} y \in H \cap H'$ *for some source variable* $x \in \mathsf{SV}(\rho) \cap \mathsf{SV}(\rho')$ *and action* b.
- $\models_G \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(J)$.

2. *If* ρ *is of the form* $\dfrac{H}{Pr\ P}$ *then there exists a finite set* J *of valid ruloids for* Q *with conclusion* $Pr\ Q$ *such that:*
   - *For every* $\rho' = \frac{H'}{Pr\ Q} \in J$, *we have:*
     - $(\mathsf{TV}(\rho') \cup \mathsf{TV}(\rho)) \cap (\mathsf{SV}(\rho) \cup \mathsf{SV}(\rho')) = \emptyset$.
     - *If* $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$, *then* $x \xrightarrow{b} y \in H \cap H'$ *for some source variable* $x \in \mathsf{SV}(\rho) \cap \mathsf{SV}(\rho')$ *and action* b.
   - $\models_G \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(J)$.

We write $P \leftrightsquigarrow_G^{RMp} Q$ *if there exists a rule-matching bisimulation* $\approx$ *relating* P *and* Q. *As before, we refer to the relation* $\leftrightsquigarrow_G^{RMp}$ *as* rule-matching bisimilarity.

Theorem 2.8.9 (Conservative extension) *Let* G *be a GSOS language. Then,* $P \hat{\leftrightarrow}_G Q$ *iff* $P \leftrightsquigarrow_G^{RMp} Q$, *for all* $P, Q \in \mathbb{T}(\Sigma_G)$.

Proof. Since G is a GSOS language, every ruloid for any term P has the form $\dfrac{H}{P \xrightarrow{a} P'}$, where the premises in H do not use predicates. So $\mathsf{hyps}(H)$ does not involve predicate formulae either. We therefore fall under clause 1 of Definition 2.8.8, which matches exactly with Definition 2.5.1 of $\hat{\leftrightarrow}_G$.

It follows from Theorem 2.8.9 that rule-matching bisimilarity is incomplete over GSOS languages with predicates, since it is not complete even within the framework of standard GSOS. Examples 2.5.4 and 2.5.5 witness incompleteness for $\leftrightsquigarrow_G^{RMp}$, too.

Theorem 2.8.10 (Soundness) *Let* G *be a GSOS language with predicates. Then, for all* $P, Q \in \mathbb{T}(\Sigma_G)$, $P \leftrightsquigarrow_G^{RMp} Q$ *implies* $\mathsf{Bisim}(G) \models P = Q$.

Proof. The proof follows the lines of the one for Theorem 2.5.3. To show that $P \leftrightsquigarrow_G^{RMp} Q$ implies $\mathsf{Bisim}(G) \models P = Q$, it is sufficient to prove that the relation $\sim$ given by

$$\sim \;=\; \{(P\sigma, Q\sigma) \mid P \leftrightsquigarrow_G^{RMp} Q,\ \sigma \text{ a closed substitution}\}$$

is a bisimulation. The addition of predicates adds no complications to the proof, which is therefore omitted.

Following the lines of the proof of Theorem 2.5.8, one obtains the following result. (The notion of disjoint extension used in the statement below is the obvious modification of the one over GSOS languages—see Definition 2.2.8—to a setting with predicates. In particular, if G' disjointly extends G then G' adds no new rules for the predicate symbols in G.)

Theorem 2.8.11 *Let G be a GSOS language with predicates.*

1. *Assume that $\approx$ is a G-robust rule-matching bisimulation over G, and let G' be a disjoint extension of G. Then $\approx$ is a rule-matching bisimulation over G'.*
2. *Let $P, Q \in \mathbb{T}(\Sigma_G)$. Assume that $P \leftrightsquigarrow_G^{RMp} Q$ can be shown by establishing a G-robust rule-matching bisimulation over G. Then $\mathsf{BISIM}(G) \models P = Q$.*

All the examples of rule-matching bisimulation in a setting with predicates we have met so far in our analysis of sound equations from the literature on process algebra are robust.

2.8.5 Examples

We now present some examples of application of rule-matching bisimilarity to a setting with predicates.

Example 2.8.12 (Sequential composition: zero element) *Consider the sequential composition operator from Example 2.8.3 on page 47. Recall that the constant a^ω is defined by the single axiom $a^\omega \xrightarrow{a} a^\omega$. This constant simply displays a infinitely many times. a^ω is a left-zero element for '$\cdot$', because its infinite behaviour is enough to preempt the execution of the right-hand argument of '$\cdot$'. Our order of business in this example is to check the law $a^\omega \cdot y \leftrightsquigarrow a^\omega$ by means of rule-matching bisimilarity. To this end, it is sufficient to show that the relation $\{(a^\omega \cdot y, a^\omega), (a^\omega, a^\omega \cdot y)\}$ is a rule-matching bisimilarity.*

In constructing the set of ruloids for $a^\omega \cdot y$, rules (seq2) and (seq3) in Example 2.8.3 play no role because the premise $x \downarrow$ is not satisfied when x is instantiated with the constant a^ω. The only ruloid generated by the ruloid theorem comes from the instantiation of rule (seq1) and it is the axiom $a^\omega \cdot y \xrightarrow{a} a^\omega \cdot y$. The set of ruloids for the constant a^ω consists of the singleton set containing the axiom defining the constant. Since the two ruloids match, meeting all the constraints of Definition 2.8.8, by the soundness theorem (Theorem 2.8.10) we can conclude that a^ω is a left zero element for $\cdot$.

Example 2.8.13 (Sequential composition: associativity) *In this example, our aim is to prove the associativity of the sequential composition operator '$\cdot$' considered in the previous example.*

Let $R[x,y,z] = x \cdot (y \cdot z)$ and $L[x,y,z] = (x \cdot y) \cdot z$, consider the symmetric closure of the relation

$$\approx \;\triangleq\; \{(R[x,y,z], L[x,y,z]) \mid x,y,z \in \mathsf{Var}\} \cup \mathcal{I}$$

where $\mathcal{I}$ denotes the identity relation over $\mathbb{T}(\Sigma_G)$.

As in the case of associativity of the sequencing operator considered in Section 2.6, for the sake of clarity, we present the ruloids of R and L and their 'matching' in the following suggestive way.

$$\frac{x \xrightarrow{a} x'}{\begin{array}{c} L \xrightarrow{a} (x' \cdot y) \cdot z \\ R \xrightarrow{a} x' \cdot (y \cdot z) \end{array}} \qquad \frac{x \downarrow, \quad y \xrightarrow{a} y'}{\begin{array}{c} L \xrightarrow{a} y' \cdot z \\ R \xrightarrow{a} y' \cdot z \end{array}} \qquad \frac{x \downarrow, \quad y \downarrow, \quad z \xrightarrow{a} z'}{\begin{array}{c} L \xrightarrow{a} z' \\ R \xrightarrow{a} z' \end{array}} \qquad \frac{x \downarrow, \quad y \downarrow, \quad z \downarrow}{\begin{array}{c} L \downarrow \\ R \downarrow \end{array}}$$

The reader can easily convince himself that any ruloid for R matches with its corresponding one for L satisfying the conditions of Definition 2.8.8, and vice versa. The relation $\approx$ is indeed a rule-matching bisimulation and, by the soundness theorem, the associativity of this sequential composition operator follows.

Example 2.8.14 (Predictable failure constant of $BPA_{0\delta}$) *In this example we focus on $BPA_{0\delta}$ of Baeten and Bergstra, [25]. The predictable failure 0 is a constant that absorbs the computation no matter where it appears within the context of the (non standard) sequential composition operator '$\odot$', i.e. 0 is a zero element for the operator '$\odot$'. (Baeten and Bergstra use $\cdot$ to denote the sequential composition operator in $BPA_{0\delta}$. We denote that operator by $\odot$ here in order to avoid any confusion with the standard sequential composition operator considered in Example 2.8.3.) The laws $x \odot 0 \leftrightarrows 0 \odot x \leftrightarrows 0$ both*

hold and our order of business in this example is to check them by means of $\leftrightarrows^{RMp}$. *The following SOS rules make use of the predicate* $\neq 0$ *that determines whether or not a process can be proved equal to* 0 *and of predicates* $\xrightarrow{a} \checkmark$ *that tell us when a process can terminate by performing an a action.*

$$\frac{x \neq 0 \quad y \neq 0}{(x \odot y) \neq 0} \qquad \frac{x \xrightarrow{a} x' \quad y \neq 0}{x \odot y \xrightarrow{a} x' \odot y} \qquad \frac{x \xrightarrow{a} \checkmark \quad y \neq 0}{x \odot y \xrightarrow{a} y}$$

Consider the equation $0 \odot x \leftrightarrows 0$. *Let* $L[x] = 0 \odot x$ *and* $R = 0$. *The set of ruloids for* R *is clearly empty, since* 0 *is defined by no rules. Also the set of ruloids for* L *is empty, due to the fact that the premises* $x \neq 0$, $x \xrightarrow{a} x'$ *and* $x \xrightarrow{a} \checkmark$ *are not satisfied when* x *is instantiated with* 0. *The contexts* L *and* R *are thus trivially equated by* $\leftrightarrows^{RMp}$. *The reader can easily realize that the scenario is similar when it comes to checking the law* $x \odot 0 \leftrightarrows 0$. *The ruloid set for* $x \odot 0$ *is again empty, due to the fact that the premise* $y \neq 0$, *contained in all of the three rules above, is not satisfied when* y *is instantiated with* 0. *Thanks to Theorem 2.8.10, this is sufficient to conclude that the constant* 0 *is both a left and a right zero element for '*$\odot$*'.*

In carrying out our study of the rule-matching bisimilarity we established some criteria in order to preserve the soundness of certain equations in disjoint extensions of a semantics specified in the GSOS rule format. The next paragraph is devoted to the discussion of some work related to this specific subject.

A remark on the robustness of behavioural equivalences. In [103], Peter Mosses, MohammadReza Mousavi and Michel Reniers address the topic of the robustness of equations on open terms. Sound equations indeed may become unsound after an operationally conservative extension, for examples see [107] , [103] and Example 2.5.6 on page 37. It is thus desirable to provide criteria under which the validity of equations is instead preserved. The main benefit is that it prevents the user from repeating proofs in the context of the new extended semantics.

We provide some criteria along this line, with Theorems 2.5.8 and 2.8.11. In this section we shall briefly discuss and compare them with the results in [103]. The authors consider the extension of the bisimilarity to open terms (there called *closed-instance bisimilarity*), the FH-bisimilarity of De Simone [52] and its mentioned sharpening due to Rensink called *hypothesis preserving bisimilarity*, HP-bisimilarity, [125] . Insofar the work presented in this chapter is concerned the results regarding the FH- and HP-bisimilarity are relevant. In [103], two results are provided:

- (1) For FH- and HP-bisimilarity, sound equations are preserved by all disjoint extensions that do not add new labels.
- (2) If a FH- and HP-bisimilarity relation does not contain pairs (s, t) of terms such that both s and t are not just variables or they are the same variable, sound equations are preserved by any arbitrary disjoint extensions.

The reader may notice that both restrictions (1) and (2) are somewhat orthogonal to the conditions proposed in Theorems 2.5.8 and 2.8.11. In particular, within the latter theorems the considered language must be sufficiently expressive, thus the restriction is on the semantics of the original language that later will be extended. In (1) the restriction is instead on the extended semantics, i.e. the new language must not add new labels. In (2) a restriction is applied instead to the form of the equations.

Results in [103] are provided for the positive fragment of GSOS, i.e. containing only positive premises. Theorem 2.5.8 applies to the full GSOS format, i.e., involving negative premises, too, and Theorem 2.8.11 lifts the same result also to the context of full GSOS languages extended with predicates. The results on preserving the soundness of equations that are presented in this chapter are thus to be considered an addendum to the ones in [103].

The interested reader is however invited to refer to [103] which contains a reasoned account to the subject.

In [102], an extended version of [103], the same authors provide also some discussion on the rule-matching bisimilarity, in particular, it is worth mentioning that they point out that Example 2.5.6 shows that the rule matching bisimilarity does not preserve soundness of equations for those disjoint extensions that even do not introduce new labels, (restriction (1)). As pointed out in [102], the main problem of the rule-matching bisimilarity when preserving the soundness of equations may be found in the accuracy of the entailment relation employed. The logic is indeed expressive enough to prove $\models (x \xrightarrow{a}) \Rightarrow (y \xrightarrow{a})$ in those languages where every term is performing an a-transition. On the one hand, this is a good plus for the method proposed, as though it succeeds in approximate the bisimilarity of open terms the closest, but on the other hand it shows that its machinery is powerful enough to detect delicate insights of the semantics at hand. These insights may often change even when simple extensions are applied.

2.9 Related and future work

The development of general methods for proving equivalences between open terms in expressive process calculi is a challenging subject that has received some attention since the early developments of the algebraic theory of processes—see, *e.g.*, the references [47, 88, 125, 52, 149] for some of the work in this area. De Simone's FH-bisimilarity [52] represents an early meaningful step towards a general account of the problem, presenting for the first time a sound bisimulation method in place of the usual definition which involves the closure under all possible substitutions. Our method relies mainly on the concepts underlying FH-bisimilarity and it is a refinement of that notion in the more expressive setting of GSOS languages. (See de Simone's 'Clock Example' discussed on page 41, where FH-bisimilarity fails while $\hat{=}$ succeeds.)

Later Rensink addressed the problem of checking bisimilarity of open terms in [125], where he presented a natural sharpening of de Simone's FH-bisimilarity. His extension of FH-bisimilarity is orthogonal to ours and provides another method to check equivalences between open terms that is more powerful than the original FH-bisimilarity. Rensink defined a new notion of bisimulation equivalence, called *hypothesis preserving bisimilarity*, that adds to FH-bisimilarity the capability to store some kind of information about the variable transitions during the computation.

To explain the import of hypothesis preserving bisimilarity we can look at Example 2.5.4. We note that $\hat{=}$ fails to establish the sound equation $h(x) = i(x)$ because at the second step of the computation some knowledge about the transitions of the closed term p substituted for x is already established (indeed, at that point we know that p performs a b-transition, since this has been tested at the first step). Nevertheless, when comparing $f(x)$ and $g(x)$, rule-matching bisimulation behaves in memoryless fashion and ignores this information. Rensink's hypothesis preserving bisimilarity takes into account the history and this is enough to overcome the difficulties in that example and analogous scenarios. Adding this feature to $\hat{=}$ would lead to a more powerful rule-matching equivalence; we leave this further sharpening for future work together with extensions of $\hat{=}$ to more expressive rule formats.

Recently, van Weerdenburg addressed the automation of soundness proofs in [149]. His approach differs from the one in [125, 52] and ours since he translates the operational semantics into a logical framework. In such a framework, rules are

encoded as logical formulae and the overall semantics turns out to be a logical theory, for which van Weerdenburg provides a sequent calculus style proof system. In the aforementioned paper, he offers some examples of equivalences from the literature that can be proved using his method in order to highlight its applicability. However, even though the ultimate aim of the research described in [149] is the automation of soundness proofs, van Weerdenburg's system presents some drawbacks. The main point is that the user is not only required to provide the operational semantics and the equation to check (together with the standard encoding of bisimilarity), but he must also provide a candidate bisimulation relation that can be used to show the validity of the equation under consideration together with all the axioms that are needed to complete the proof. The user is supposed thus to have a clear understanding of what the proof is going to look like. This seems to be a general and inescapable drawback when approaching the problem of checking equations through a translation into a logical system.

Despite the aforementioned slight drawback, the approach proposed by van Weerdenburg is, however, very interesting and complements the proposals that are based on the ideas underlying de Simone's FH-bisimilarity, including ours. We believe that an adequate solution to the problem of automating checks for the validity of equations in process calculi will be based on a combination of bisimulation-based and logical approaches.

A related line of work is the one pursued in, *e.g.*, the papers [4, 22, 49, 110]. Those papers present rule formats that guarantee the soundness of certain algebraic laws over a process language 'by design', provided that the SOS rules giving the semantics of certain operators fit that format. This is an orthogonal line of investigation to the one reported in this article. As a test case for the applicability of our rule-based bisimilarity, we have checked that the soundness of all the equations guaranteed to hold by the commutativity format from [110] can be shown using 🎩. We are carrying out similar investigations for the rule formats proposed in [4, 49].

Another avenue for future research we are actively pursuing is the search for more, and more general, examples of partial completeness results for rule-matching bisimulation over GSOS and de Simone languages. Indeed, the partial completeness results we present in Section 2.7 are just preliminary steps that leave substantial room for improvement. Last, but not least, we are about to start working on an implementation of a prototype checker for rule-matching bisimilarity.

2.10 Proof of Theorem 2.5.3

Definition 2.10.1 *Let f and g be functions, and S a subset of both their domains. Then $f = g$ on S means, $\forall x \in S. f(x) = g(x)$.*

The following basic lemma about $\models$ will be useful in what follows.

Lemma 2.10.2 *Let σ, σ' be closed substitutions and $S \subseteq \mathsf{Var}$. Assume that $\sigma = \sigma'$ on S. Then, for all initial transition formulae F such that $vars(F) \subseteq S$,*

$$\rightarrow_G, \sigma \models F \Leftrightarrow \rightarrow_G, \sigma' \models F \ .$$

We are now ready to embark on the proof of Theorem 2.5.3. To show that $P \mathrel{\hat{\leftrightarrow}}_G Q$ implies $\mathsf{Bisim}(G) \models P = Q$, it is sufficient to prove that the relation $\sim$ given by

$$\sim \ = \{(P\sigma, Q\sigma) \mid P \mathrel{\hat{\leftrightarrow}}_G Q, \ \sigma \text{ a closed substitution}\}$$

is a bisimulation. First of all, note that $\sim$ is symmetric as $\mathrel{\hat{\leftrightarrow}}_G$ is.

Assume then that $P\sigma \sim Q\sigma$, and that $P\sigma \xrightarrow{a} p$. We will show that $Q\sigma \xrightarrow{a} q$ for some q such that $p \sim q$. By the ruloid theorem, $P\sigma \xrightarrow{a} p$ because there is some ruloid $\rho = \frac{H}{P \xrightarrow{a} P'} \in R_G(P)$ enabling this transition, and some substitution σ' such that:

1. $\sigma' \models H$,
2. $P\sigma' \equiv P\sigma$, and
3. $P'\sigma' \equiv p$.

Note that, by 1, we have that $\sigma' \models \mathsf{hyps}(\rho)$. Moreover, by 2, $\sigma = \sigma'$ on $vars(P)$. Thus, as the variables in $\mathsf{hyps}(\rho)$ are included in $vars(P)$, Lemma 2.10.2 gives $\sigma \models \mathsf{hyps}(\rho)$.

As $P \mathrel{\hat{\leftrightarrow}} Q$, there exists a set J of valid ruloids for Q which satisfies the conditions in Definition 2.5.1. In particular, by condition 1c, we have that no target variable in a ruloid $\rho' \in J$ occurs as a source variable in ρ and, vice versa, no source variable in a ruloid $\rho' \in J$ occurs as a target variable in ρ.

Our aim now is to find a move from $Q\sigma$ matching the transition $P\sigma \equiv P\sigma' \xrightarrow{a} P'\sigma' \equiv p$. To this end, we will construct a substitution τ and find a ruloid $\rho' = \frac{H'}{Q \xrightarrow{a} Q'} \in J$

such that

$$\sigma = \tau \text{ on } vars(P) \cup vars(Q) \tag{2.11}$$

$$\tau \models H' \tag{2.12}$$

$$\sigma' = \tau \text{ on } \mathsf{TV}(\rho) \tag{2.13}$$

Note that (2.11) implies that $P\sigma \equiv P\sigma' \equiv P\tau$ and $Q\sigma \equiv Q\tau$. Moreover, (2.11) and (2.13) imply $\sigma' = \tau$ on $vars(\rho)$, and thus that $p \equiv P'\sigma' \equiv P'\tau$. Condition (2.12) will make sure that the selected ruloid fires under the substitution τ.

To this end, consider the substitution σ'' given by

$$\sigma''(x) \quad = \quad \begin{cases} \sigma(x) & x \in vars(P) \cup vars(Q) \\ \sigma'(x) & \text{otherwise} \end{cases}$$

Note that, as $\mathsf{TV}(\rho) \cap vars(Q) = \emptyset$, $\sigma'' = \sigma'$ on $vars(\rho)$. Thus $\sigma'' \models H$ and, *a fortiori*, $\sigma'' \models \mathsf{hyps}(\rho)$. So, by part 2 of the definition of rule-matching bisimulation, we have $\sigma'' \models \mathsf{hyps}(J)$.

As $\mathsf{hyps}(J) = \bigvee_{\rho' \in J} \mathsf{hyps}(\rho')$, we have $\sigma'' \models \mathsf{hyps}(\rho')$ for some $\rho' = \frac{H'}{Q \xrightarrow{a} Q'} \in J$. As $\sigma'' \models \mathsf{hyps}(\rho')$, there is a substitution τ' with $\tau' = \sigma''$ on $vars(Q)$ such that $\tau' \models H'$. (Note that τ' need not be consistent with σ'' on $\mathsf{TV}(\rho)$.)

Let now

$$\tau(x) = \begin{cases} \sigma''(x) & x \in vars(\rho) \cup vars(Q) \\ \tau'(x) & \text{otherwise} \end{cases} \tag{2.14}$$

Note that $\tau = \sigma = \sigma'' = \tau'$ on $vars(Q)$.

We claim that $\tau \models H'$. To see that this is indeed the case, we consider each hypothesis in H' in turn:

$x \xrightarrow{b} y$, $y \in \mathsf{TV}(\rho)$: In this case, by part 1d of Definition 2.5.1, we have $x \xrightarrow{b} y \in H \cap H'$; that is, the same transition formula is an antecedent of both rules. As $\sigma'' \models H$, we have $\sigma''(x) \xrightarrow{b} \sigma''(y)$. By definition of τ, we have $\tau(x) = \sigma''(x)$ and $\tau(y) = \sigma''(y)$; hence $\tau \models x \xrightarrow{b} y$.

$x \xrightarrow{b} y$, $y \notin \mathsf{TV}(\rho)$: In this case, $x \in \mathsf{SV}(\rho') = vars(Q)$, so $\tau(x) = \tau'(x)$. As $y \notin vars(\rho) \cup vars(Q)$, we have that $\tau(y) = \tau'(y)$. As, by construction, $\tau' \models H'$, we have $\tau(x) \equiv \tau'(x) \xrightarrow{b} \tau'(y) \equiv \tau(y)$, and hence $\left\langle \tau \models x \xrightarrow{b} y \right\rangle$ as desired.

$x \overset{b}{\nrightarrow}$: In this case, we have $x \in \mathsf{SV}(\rho') = \mathit{vars}(Q)$, and so the substitutions τ and τ' give the same value for x. As $\tau' \models H'$, $\tau'(x) \overset{b}{\nrightarrow}$, whence $\tau(x) \overset{b}{\nrightarrow}$ as desired.

Hence $\tau \models H'$, and so ρ' fires on τ. That is, we have $Q\sigma \equiv Q\tau \overset{a}{\to} Q'\tau$. We claim that $Q'\tau$ is the closed term q we were looking for. In fact, $P'\sigma' \equiv P'\tau$, as $\sigma' = \sigma'' = \tau$ on $\mathit{vars}(\rho)$. This shows that $\sim$ is a bisimulation relation.

2.11 Proof of Theorem 2.7.2

The 'if' direction follows from Theorem 2.5.3. To prove that $\mathsf{Bisim}(G) \models P = Q$ implies $P \leftrightarrows Q$, it is sufficient to show that the relation

$$\approx \;=\; \{(P, Q) \mid \mathsf{Bisim}(G) \models P = Q, \;\; P, Q \text{ persistent}\}$$

is a rule-matching bisimulation. To this end, let P, Q be persistent contexts such that $\mathsf{Bisim}(G) \models P = Q$, and let $\rho = \frac{H}{P \overset{a}{\to} P} \in R_G(P)$. By Theorem 2.3.2, we may safely assume that ρ is such that $\mathsf{TV}(\rho) \cap \mathit{vars}(Q) = \emptyset$. We want to find a finite set J_ρ of valid ruloids for Q satisfying the conditions of the definition of $\leftrightarrows$. We will now show how to construct such a J_ρ.

Let σ be a closed substitution such that $\to_G, \sigma \models H$. Then, as ρ is sound, we have that $P\sigma \overset{a}{\to} P\sigma$. As $\mathsf{Bisim}(G) \models P = Q$, it follows that $P\sigma \leftrightarrows Q\sigma$. Hence there exists a process q such that $Q\sigma \overset{a}{\to} q$ and $P\sigma \leftrightarrows q$. By Theorem 2.3.2, this is because there exist a ruloid $\rho'_\sigma = \frac{H'}{Q \overset{a}{\to} Q} \in R_G(Q)$ and a substitution τ_σ such that

- $\tau_\sigma \models H'$, and
- $Q\sigma \equiv Q\tau_\sigma \equiv q$.

Note that $Q\sigma \equiv Q\tau_\sigma$ implies that $\sigma = \tau_\sigma$ on $\mathit{vars}(Q)$. By suitably renaming the variables in $\mathsf{TV}(\rho'_\sigma)$, it is now easy to construct a valid ruloid $\hat{\rho}_\sigma = \frac{\hat{H}'}{Q \overset{a}{\to} Q}$ for Q, and a modified substitution $\hat{\tau}_\sigma$ such that

- $\mathsf{TV}(\hat{\rho}_\sigma) \cap \mathsf{TV}(\rho) = \emptyset$ and $\mathsf{TV}(\hat{\rho}_\sigma) \cap \mathit{vars}(P) = \emptyset$,
- $\hat{\tau}_\sigma \models \hat{H}'$, and
- $Q\sigma \equiv Q\tau_\sigma = Q\hat{\tau}_\sigma$.

Thus, for each closed substitution σ such that $\to_G, \sigma \models H$, we can construct a valid ruloid $\hat{\rho}_\sigma$ and a closed substitution $\hat{\tau}_\sigma$ with the above properties.

Take now $J_\rho = \{\hat{\rho}_\sigma \mid \rightarrow_G, \sigma \models H\}$. Note that, by the ruloid theorem, we can assume, without loss of generality, that J_ρ is finite. We claim that J_ρ is a set of ruloids matching the conditions in Definition 2.5.1. In fact, each $\hat{\rho}_\sigma$ has action a, $P \approx Q$ by construction, and conditions 1c and 1d are trivially met by construction. Moreover, we have that $\models \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(J_\rho)$. Assume, in fact, that $\rightarrow_G, \sigma \models \mathsf{hyps}(\rho)$. Then we know that, by construction, $\hat{\tau}_\sigma \models \mathsf{hyps}(\hat{\rho}_\sigma)$, and that $\sigma = \hat{\tau}_\sigma$ on $vars(Q)$. As $vars(\mathsf{hyps}(\hat{\rho}_\sigma)) \subseteq vars(Q)$, Lemma 2.10.2 gives $\sigma \models \mathsf{hyps}(\hat{\rho}_\sigma)$. As $\hat{\rho}_\sigma \in J_\rho$, we then have that $\sigma \models \mathsf{hyps}(J_\rho)$. Hence $\approx$ is indeed a rule-matching bisimulation.

2.12 Proof of Theorem 2.7.4

Let G be a non-inheriting GSOS language that, for each $P \in \mathbb{T}(\Sigma_G)$ and $c \in \mathsf{Act}$, contains at most one ruloid for P having $c \in \mathsf{Act}$ as action. Let G' be the disjoint extension of G obtained by adding to G the operations and rules of the language BCCSP [146, 96] with Act as set of actions. Let P and Q be terms over Σ_G, and assume that $\mathsf{Bisim}(G') \models P = Q$. We aim at showing that $P \leftrightarrows_{G'} Q$. To prove our claim, it suffices only to show that the relation

$$\approx \;\; = \{(P, Q) \mid \mathsf{Bisim}(G') \models P = Q,\ P, Q \in \mathbb{T}(\Sigma_G)\}$$

is a rule-matching bisimulation. Note, first of all, that $\approx$ is symmetric because so is $\leftrightarrows_{G'}$.

Assume now that $P \leftrightarrows_{G'} Q$ and $P, Q \in \mathbb{T}(\Sigma_G)$. We proceed to argue that the conditions in Definition 2.5.1 are met by $\approx$. Using Theorem 2.3.2, we start by constructing the set of ruloids for P and Q in such a way that $(\mathsf{TV}(\rho') \cup \mathsf{TV}(\rho)) \cap (\mathsf{SV}(\rho) \cup \mathsf{SV}(\rho')) = \emptyset$. This meets condition 1c in Definition 2.5.1.

Let $\rho = \dfrac{H}{P \xrightarrow{a} P'}$ be a ruloid for P, which we may assume is not junk. We show that there is a ruloid $\rho' = \dfrac{H'}{Q \xrightarrow{a} Q'}$ such that

1. $P' \approx Q'$,
2. if $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$, then $x \xrightarrow{b} y \in H \cap H'$ for some source variable $x \in \mathsf{SV}(\rho) \cap \mathsf{SV}(\rho')$ and action b, and
3. $\models_{G'} \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(\rho')$.

Note first of all that, by the proviso of the theorem, the set of ruloids for Q with action a must be a singleton. Indeed, that set cannot be empty since ρ is a ruloid for P whose premises can be satisfied by some closed substitution σ, as ρ is not junk. This means that $P\sigma$ affords an a-labelled transition, but $Q\sigma$ would not. This contradicts our assumption that $P \leftrightarrows_{G'} Q$.

Let $\rho' = \dfrac{H'}{Q \xrightarrow{a} Q'}$ be the only ruloid for Q with action a. We proceed to establish the above three conditions in turn.

1. We show that $P' \approx Q'$. To this end, we begin by observing that $P', Q' \in \mathbb{T}(\Sigma_G)$ because G' is a disjoint extension of G and $P, Q \in \mathbb{T}(\Sigma_G)$. We are therefore left to prove that $P' \leftrightarrows_{G'} Q'$. Assume, towards a contradiction, that $P' \underline{\leftrightarrows}_{G'} Q'$. This means that there is a closed substitution σ', mapping variables to terms in $\mathrm{T}(\Sigma_{G'})$, such that $P'\sigma' \underline{\leftrightarrows}_{G'} Q'\sigma'$. Our order of business will now be to use σ' to construct a closed substitution σ such that $P\sigma \underline{\leftrightarrows}_{G'} Q\sigma$, whose existence contradicts our assumption that $\mathsf{Bisim}(G') \models P = Q$.

   Define σ thus:

$$\sigma(x) \;=\; \begin{cases} \sum \left\{ b.\sigma'(x') \mid (x \xrightarrow{b} x') \in H \right\} & \text{if } x \in \mathit{vars}(P), \\ \sigma'(x) & \text{otherwise.} \end{cases}$$

   Note that terms of the form $\sum \left\{ b.\sigma'(x') \mid (x \xrightarrow{b} x') \in H \right\}$ are in $\mathrm{T}(\Sigma_{G'})$ since $\Sigma_{G'}$ includes the signature of BCCSP. As usual, an empty sum stands for $\mathbf{0}$.

   We claim that $\rightarrow_{G'}, \sigma \models H$. Indeed, suppose that $(x \xrightarrow{b} x') \in H$. Then, since $x \notin \mathit{vars}(P)$,

$$\sigma(x) \xrightarrow{b} \sigma'(x') = \sigma(x') \enspace .$$

   Assume now $x \overset{b}{\nrightarrow} \in H$. Since ρ is not junk, H contains no formula of the form $(x \xrightarrow{b} x')$. Therefore, by definition, $\sigma(x) \overset{b}{\nrightarrow}$. It follows that ruloid ρ fires under substitution σ and therefore

$$P\sigma \xrightarrow{a} P'\sigma = P'\sigma' \enspace .$$

   (The equality $P'\sigma = P'\sigma'$ holds because G is non-inheriting and therefore no variable occurring in P' occurs also in P.) Since $P\sigma \leftrightarrows_{G'} Q\sigma$ and ρ' is the only a-labelled ruloid for Q, there is a transition

$$Q\sigma \xrightarrow{a} Q'\sigma = Q'\sigma' \leftrightarrows_{G'} P'\sigma' \enspace .$$

(Again, the equality $Q'\sigma = Q'\sigma'$ holds because G is non-inheriting and no target variable occurring in ρ' occurs also in Q.) This contradicts our assumption that $P'\sigma' \underline{\leftrightarrow}_{G'} Q'\sigma'$. We may therefore conclude that $P' \leftrightarrow_{G'} Q'$, as desired.

2. We show that if $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$, then $x \xrightarrow{b} y \in H \cap H'$ for some source variable $x \in \mathsf{SV}(\rho) \cap \mathsf{SV}(\rho')$ and action b.

   Assume that $y \in \mathsf{TV}(\rho) \cap \mathsf{TV}(\rho')$. Let $x \xrightarrow{b} y$ be the only premise in H with target y and let $z \xrightarrow{c} y$ be the only premise in H' with target y. We shall now show that $b = c$ and $x = z$. Indeed, assume first, towards a contradiction, that $x \neq z$. Using this assumption, we shall construct a closed substitution σ such that $P\sigma \underline{\leftrightarrow}_{G'} Q\sigma$, which contradicts $\mathsf{Bisim}(G') \models P = Q$.

   Let X be the collection of target variables w' in ρ and ρ' that are different from y and for which there is a premise of the form $w \xrightarrow{c} w'$ in $H \cup H'$. For each $w \in vars(P)$ and action d, let the closed term p^d_w be given by:

$$p^d_w = \begin{cases} d.d.\mathbf{0} & \exists w'.\ (w \xrightarrow{d} w') \in H, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{2.15}$$

   Define σ thus:

$$\sigma(w) = \begin{cases} \sum \left\{ d.\mathbf{0} \mid \exists w'.\ (w \xrightarrow{d} w') \in H \right\} + p^c_w & \text{if } w \in vars(P) - \{z\}, \\ \sum \left\{ d.\mathbf{0} \mid d \neq c \wedge \exists w'.\ (z \xrightarrow{d} w') \in H \right\} + c.c.\mathbf{0} & \text{if } w = z, \\ c.\mathbf{0} & \text{if } w \in X, \text{ and} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

   It is not hard to see that σ satisfies H, but not the formula $z \xrightarrow{c} y$. Therefore, ruloid ρ fires under substitution σ, but ρ' does not. Reasoning as above, this yields that $P\sigma$ and $Q\sigma$ are not bisimilar as desired.

   So, $x \xrightarrow{b} y \in H$ and $x \xrightarrow{c} y \in H'$. We shall now argue that $b = c$, completing the proof for this case. Assume, towards a contradiction, that $b \neq c$. As above, using this assumption, we shall construct a closed substitution σ such that $P\sigma \underline{\leftrightarrow}_{G'} Q\sigma$, which contradicts $\mathsf{Bisim}(G') \models P = Q$.

   Let Y be the collection of target variables w' in ρ for which there is a premise of the form $w \xrightarrow{b} w'$ in H. In particular, $y \in Y$. Define σ thus, where the term

p^b_w is defined as in (2.15):

$$\sigma(w) \;=\; \begin{cases} \sum\left\{d.\mathbf{0} \mid d \neq b \wedge \exists w'.\ (w \xrightarrow{d} w') \in H\right\} + p^b_w & \text{if } w \in vars(P), \\ b.\mathbf{0} & \text{if } w \in Y, \text{ and} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

It is not hard to see that σ satisfies H, but not the formula $x \xrightarrow{c} y$. Therefore, ruloid ρ fires under substitution σ, but ρ' does not. Reasoning as above, this yields that $P\sigma$ and $Q\sigma$ are not bisimilar as desired. We may finally conclude that $b = c$ and therefore that $x \xrightarrow{b} y \in H \cap H'$, which was to be shown.

3. We show that $\models_{G'} \mathsf{hyps}(\rho) \Rightarrow \mathsf{hyps}(\rho')$. To this end, assume, towards a contradiction, that there is a closed substitution σ that satisfies $\mathsf{hyps}(\rho)$, but not $\mathsf{hyps}(\rho')$. We shall use σ to construct a substitution σ' such that $P\sigma' \mathrel{\underline{\leftrightarrow}}_{G'} Q\sigma'$, contradicting our assumption that $P \leftrightarrow_{G'} Q$.

   Define σ' thus:

$$\sigma'(x) \;=\; \begin{cases} \sum\left\{b.\mathbf{0} \mid \exists p.\ \sigma(x) \xrightarrow{b} p\right\} & \text{if } x \in \mathsf{SV}(\rho) \cup \mathsf{SV}(\rho'), \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

   We claim that $\rightarrow_{G'}, \sigma \models H$. Indeed, suppose that $(x \xrightarrow{b} x') \in H$. Then, $x \xrightarrow{b}$ is a conjunct of $\mathsf{hyps}(\rho)$ and $x \in \mathsf{SV}(\rho)$. Since σ satisfies $\mathsf{hyps}(\rho)$, it follows that $\sigma(x) \xrightarrow{b} p$ for some closed term p. By the definition of σ' and the fact that $x' \notin \mathsf{SV}(\rho) \cup \mathsf{SV}(\rho')$,

$$\sigma'(x) \xrightarrow{b} \mathbf{0} = \sigma'(x') \ .$$

   Assume now $x \xrightarrow{b}\!\!\!\!\!\!\in H$. Then, $x \xrightarrow{b}\!\!\!\!\!\!\rightarrow$ is a conjunct of $\mathsf{hyps}(\rho)$ and $x \in \mathsf{SV}(\rho)$. Since σ satisfies $\mathsf{hyps}(\rho)$, it follows that $\sigma(x) \xrightarrow{b}\!\!\!\!\!\!\rightarrow$. By the definition of σ', we have $\sigma'(x) \xrightarrow{b}\!\!\!\!\!\!\rightarrow$ and we are done. It follows that ruloid ρ fires under substitution σ' and therefore

$$P\sigma' \xrightarrow{a} P'\sigma' \ .$$

   We shall now argue that $\rightarrow_{G'}, \sigma' \not\models H'$. This means that $Q\sigma'$ does not afford an a-labelled transition, and therefore $P\sigma' \mathrel{\underline{\leftrightarrow}}_{G'} Q\sigma'$, as desired.

   Since σ does not satisfy $\mathsf{hyps}(\rho')$, there is a conjunct of that formula that is not satisfied by σ. We proceed with the proof of our claim by considering the two possible forms this conjunct may take.

- Assume that σ does not satisfy a conjunct of the form $x \xrightarrow{b}$ in $\mathsf{hyps}(\rho')$. This means that $\sigma(x) \overset{b}{\nrightarrow}$ and that $(x \xrightarrow{b} x') \in H'$ for some x'. Note that $x \in \mathsf{SV}(\rho')$. Therefore $\sigma'(x) \overset{b}{\nrightarrow}$, by the definition of σ', and $\rightarrow_{G'}, \sigma' \not\models H'$, as claimed.
- Assume that σ does not satisfy a conjunct of the form $x \overset{b}{\nrightarrow}$ in $\mathsf{hyps}(\rho')$. This means that $\sigma(x) \xrightarrow{b} p$ for some closed term p and that $(x \overset{b}{\nrightarrow}) \in H'$. Note that $x \in \mathsf{SV}(\rho')$. Therefore $\sigma'(x) \xrightarrow{b} \mathbf{0}$, by the definition of σ', and $\rightarrow_{G'}, \sigma' \not\models H'$, as claimed.

This completes the proof.

Chapter 3

Rule Formats for Zero and Unit Elements

In a world of 1s and 0s... are you a zero, or The One?
From the movie *Matrix*.

3.1 Introduction

In the last three decades, Structural Operational Semantics (SOS), see, e.g., [18, 109, 119, 65], has been shown to be a powerful way to specify the semantics of programming and specification languages. In this approach to semantics, languages can be given a clear behaviour in terms of states and transitions, where the collection of transitions is specified by means of a collection of syntax-driven inference rules. Based on this semantics in terms of state transitions, we often want to prove general algebraic laws about the languages, which describe semantic properties of the various operators they involve modulo the notion of behavioural equivalence or preorder of interest. For example, the reader may think about the field of process algebra, where it is important to check whether certain operators are, say, commutative and associative.

This chapter aims at contributing to an ongoing line of research whose goal is to ensure the validity of algebraic properties *by design*, using the so called *SOS rule formats* [21]. Results in this research area roughly state that if the specification of (parts of) the operational semantics of a language has a certain form then some semantic property is guaranteed to hold. The literature on SOS provides rule

formats for basic algebraic properties of operators such as commutativity [110], associativity [49] and idempotence [4]. The main advantage of this approach is that one is able to verify the desired property by syntactic checks that can be mechanized. Moreover, it is interesting to use rule formats for establishing semantic properties since results so obtained apply to a broad class of languages.

Recently, in [22] the authors provided a rule format guaranteeing another basic algebraic property not addressed before: the existence of left and right unit elements for operators. In this chapter, we follow the work presented in [22] and we develop some rule formats guaranteeing instead that certain constants act as left or right zero elements for a set of binary operators. Namely, a function f has a left (respectively, right) zero element c, modulo some notion of behavioural equivalence, whenever the equation $f(c, x) = c$ (respectively, $f(x, c) = c$) holds. A constant c satisfying the above equation(s) is also said to be *absorbing* for the operator f.

A classical example of a left zero element within the realm of process algebra is provided by the constant δ, for deadlock, from BPA [37], which satisfies the laws:

$$\delta \cdot x = \delta \quad \text{and} \quad \delta \between x = \delta \ ,$$

where '$\cdot$' and '$\between$' stand for sequential composition and left merge, respectively.

The first format we provide follows the techniques developed in [22] and is of a syntactic nature. However, even though we show how several classical examples from the literature indeed fit the format, there are some basic, but somewhat more exotic, zero elements that cannot be handled by the proposed format.

We show nevertheless that we can reformulate our zero-element format within the GSOS languages of Bloom, Istrail and Meyer [44], by using a modest amount of 'semantic reasoning'. In particular, we benefit from the logic of transition formulae developed by me, Luca Aceto and Anna Ingolfsdottir in [7], which is tailored for reasoning about the satisfiability of premises of GSOS rules.

The final part of the chapter is devoted to applying the design ideas underlying the GSOS-based format for left and right zero elements to reformulate the format for left and right unit elements from [22]. The resulting format turns out to be incomparable in power to the original one, but it is expressive enough to check all the examples discussed in [22].

Mechanizing the rule formats in a tool-set is a long-term goal of research on SOS rule formats. We believe that the GSOS-based rule formats we present in this chapter are strong candidates for mechanization insofar as zero and unit elements are concerned.

Roadmap of the chapter Section 3.2 repeats some standard definitions from the theory of SOS. Section 3.3 provides the first format for left and right zero elements and Section 3.4 shows how several examples of left and right zero elements from the literature fit the format. In Section 3.5 we point out the main drawbacks of the format and in Section 3.6 we reformulate it within the GSOS format using the aforementioned logic of transition formulae. In Section 3.7 we provide a rule format for unit elements adapting the ideas from Section 3.6. We conclude the chapter with an overview of its main contributions in Section 3.8. In order to increase the readability of the main body of the chapter, the proofs of the main technical results have been collected in sections that follow Section 3.8.

3.2 Preliminaries

In this section we recall some standard definitions from the theory of SOS. We refer the readers to, e.g., [18] and [109] for more information.

3.2.1 Transition system specifications and bisimilarity

Definition 3.2.1 (Signatures, terms and substitutions) *We let V denote an infinite set of variables and use $x, x', x_i, y, y', y_i, \ldots$ to range over elements of V. 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 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, possibly subscripted and/or superscripted, 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., terms that contain no variables. We use $p, q, p', p_i, \ldots$ 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 and write* $\sigma(t)$ *for the result of applying the substitution* σ *to the term* t. *If the range of a substitution lies in* $\mathbb{C}(\Sigma)$, *we say that it is a* closed substitution.

Definition 3.2.2 (Transition system specification) *A* transition system specification *(TSS) is a triple* $(\Sigma, \mathcal{L}, D)$ *where*

- Σ *is a signature.*
- $\mathcal{L}$ *is a set of labels (or actions) ranged over by* a, b, l. *If* $l \in \mathcal{L}$, *and* $t, t' \in \mathbb{T}(\Sigma)$ *we say that* $t \xrightarrow{l} t'$ *is a* positive transition formula *and* $t \xnrightarrow{l}$ *is a* negative transition formula. *A transition formula (or just formula), typically denoted by* ϕ *or* ψ, *is either a negative transition formula or a positive one.*
- D *is a set of* deduction rules, *i.e., tuples 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 write vars(r) *to denote the set of variables appearing in a deduction rule* r. *We say that a formula or a deduction rule is* closed *if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way. For a rule* r *and a substitution* σ, *the rule* $\sigma(r)$ *is called a substitution instance of* r. *A set of positive closed formulae is called a* transition relation.

We often refer to a positive transition formula $t \xrightarrow{l} t'$ as a *transition* with t being its *source*, l its *label*, and t' its *target*. A deduction rule (Φ, ϕ) is typically written as $\frac{\Phi}{\phi}$. An *axiom* is a deduction rule with an empty set of premises. We call a deduction rule *f-defining* when the outermost function symbol appearing in the source of its conclusion is f.

In this chapter, for each constant c, *we assume that each* c*-defining deduction rule is an axiom of the form* $c \xrightarrow{l} p$ *for some label* l *and closed term* p. This is not a real restriction since all practical cases we know of do actually satisfy this property. For GSOS languages, which are defined shortly and used in later sections of this chapter, this restriction is automatically satisfied.

The meaning of a TSS is defined by the following notion of least three-valued stable model. To define this notion, we need two auxiliary definitions, namely provable transition rules and contradiction, which are given below.

Definition 3.2.3 (Provable transition rules) *A closed deduction rule is called a* transition rule *when it is of the form* $\frac{N}{\phi}$ *with* N *a set of* negative formulae. *A TSS* $\mathcal{T}$ proves

$\frac{N}{\phi}$, *denoted by* $\mathcal{T} \vdash \frac{N}{\phi}$, *when there is a well-founded upwardly branching tree with closed formulae as nodes and of which*

- *the root is labelled by* ϕ*;*
- *if a node is labelled by* ψ *and the labels of the nodes directly above it form the set* K *then:*
  - ψ *is a negative formula and* $\psi \in N$*, or*
  - ψ *is a positive formula and* $\frac{K}{\psi}$ *is a substitution instance of a deduction rule in* $\mathcal{T}$*.*

Definition 3.2.4 (Contradiction and entailment) *The formula* $t \xrightarrow{l} t'$ *is said to* contradict $t \overset{l}{\nrightarrow}$*, and vice versa. For two sets* Φ *and* Ψ *of formulae,* Φ contradicts Ψ *when there is a* $\phi \in \Phi$ *that contradicts a* $\psi \in \Psi$*. We write* $\Phi \vDash \Psi$ *when* Φ *does not contradict* Ψ*.*

A formula ϕ entails ψ *when there is a substitution* σ *such that* $\sigma(\phi) \equiv \psi$*. A set* Φ *entails a set* Ψ *of formulae, when there exists a substitution* σ *such that, for each* $\psi \in \Psi$*, there exists a* $\phi \in \Phi$ *such that* $\sigma(\phi) \equiv \psi$*. In other words,* Φ *entails* Ψ *if there is a substitution* σ *such that* $\Psi \subseteq \{\sigma(\phi) \mid \phi \in \Phi\}$*.*

It immediately follows from the above definition that contradiction is a symmetric relation on (sets of) formulae. We now have all the necessary ingredients to define the semantics of TSSs in terms of three-valued stable models.

Definition 3.2.5 (Three-valued stable model) *A pair* (C, U) *of disjoint sets of positive closed transition formulae is called a* three-valued stable model *for a TSS* $\mathcal{T}$ *when the following conditions hold:*

- *for each* $\phi \in C$*, there is a set* N *of negative formulae such that* $\mathcal{T} \vdash \frac{N}{\phi}$ *and* $C \cup U \vDash N$*, and*
- *for each* $\phi \in U$*, there is a set* N *of negative formulae such that* $\mathcal{T} \vdash \frac{N}{\phi}$ *and* $C \vDash N$*.*

C *stands for* Certainly *and* U *for* Unknown*; the third value is determined by the formulae not in* $C \cup U$*. The* least *three-valued stable model is a three-valued stable model that is the least one with respect to the ordering on pairs of sets of formulae defined as* $(C, U) \leq (C', U')$ *iff* $C \subseteq C'$ *and* $U' \subseteq U$*. We say that* $\mathcal{T}$ *is* complete *when for its least three-valued stable model it holds that* $U = \emptyset$*. In a complete TSS, we say that a closed substitution* σ *satisfies a set of formulae* Φ *if* $\sigma(\phi) \in C$*, for each positive formula* $\phi \in \Phi$*, and* $C \vDash \sigma(\phi)$*, for each negative formula* $\phi \in \Phi$*. If a TSS is complete, we often also write* $p \xrightarrow{l} p'$ *in lieu of* $(p \xrightarrow{l} p') \in C$*.*

Definition 3.2.6 (Bisimulation and bisimilarity [96, 115]) *Let $\mathcal{T}$ be a transition system specification with signature Σ and label set $\mathcal{L}$. A relation $\mathcal{R} \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* bisimulation relation *if and only if $\mathcal{R}$ is symmetric and, for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ and $l \in \mathcal{L}$,*

$$(p_0 \mathcal{R} p_1 \wedge \mathcal{T} \vdash p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (\mathcal{T} \vdash p_1 \xrightarrow{l} p_1' \wedge p_0' \mathcal{R} p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called bisimilar, *denoted by $p_0 \leftrightarroweq p_1$, when there exists a bisimulation relation $\mathcal{R}$ such that $p_0 \mathcal{R} p_1$.*

Bisimilarity is extended to open terms by requiring that $s, t \in \mathbb{T}(\Sigma)$ are bisimilar when $\sigma(s) \leftrightarroweq \sigma(t)$ for each closed substitution $\sigma : V \to \mathbb{C}(\Sigma)$.

In Sections 3.6–3.7 we focus on the GSOS format of Bloom, Istrail and Meyer [44], whose definition is given below.

Definition 3.2.7 (GSOS rule) *Suppose Σ is a signature. A* GSOS rule *r over Σ is a rule of the form:*

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} | 1 \leq j \leq m_i \right\} \ \cup \ \bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{b_{ik}} | 1 \leq k \leq n_i \right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} t} \tag{3.1}$$

where all the variables are distinct, $m_i, n_i \geq 0$, a_{ij}, b_{ik}, and c are actions from a finite set, f is a function symbol from Σ with arity l, and t is a term in $\mathbb{T}(\Sigma)$ such that $vars(t) \subseteq \{x_1, \ldots, x_l\} \cup \{y_{ij} \mid 1 \leq i \leq l, 1 \leq j \leq m_i\}$.

Definition 3.2.8 *A* GSOS language *is a triple $G = (\Sigma_G, \mathcal{L}, R_G)$, where Σ_G is a finite signature, $\mathcal{L}$ is a finite set of action labels and R_G is a finite set of GSOS rules over Σ_G.*

3.2.2 Predicates

Several of the examples of (left and right) zero and unit elements we will discuss in the remainder involve operators whose SOS semantics is best given using predicates as well as transition relations. For the sake of completeness, we therefore proceed to introduce the notion of TSS extended with predicates.

Definition 3.2.9 (Predicates) *Given a set $\mathcal{P}$ of predicate symbols, $P\ t$ is a* positive predicate formula *and $\neg P\ t$ is a* negative predicate formula, *for each $P \in \mathcal{P}$ and $t \in \mathbb{T}(\Sigma)$. We call t the* source *of both predicate formulae. In the extended setting, a (positive, negative)* formula *is either a (positive, negative) transition formula or a*

(positive, negative) predicate formula. The notions of deduction rule, TSS, provable transition rules and three-valued stable models are then naturally extended by adopting the more general notion of formulae. The label of a deduction rule is either the label of the transition formula or the predicate symbol of the predicate formula in its conclusion.

The definition of bisimulation is extended to a setting with predicates in the standard fashion. In particular, bisimilar terms must satisfy the same predicates.

3.3 Rule format

In this section we provide a rule format guaranteeing that certain constants act as left or right zero elements for a set of binary operators. To this end we employ a variation on the technique developed by some of the authors in [22] for left or right unit elements.

As in [22], we make use of an equivalence relation between terms called *zero-context equivalence*, which is the counterpart of the unit-context equivalence from [22]. Intuitively if c is a left zero element for an operator f and c is also a right zero element for g, then the terms $f(c, t_1)$ and $g(t_2, c)$ are both zero-context equivalent to c and zero-context equivalent to each other.

In the following formal definition of zero-context equivalence, it is useful to consider $(f, c) \in L$ as stating that 'c acts as a left zero element for the operator f' and analogously $(f, c) \in R$ indicates that the constant c is a right zero element for f.

Definition 3.3.1 (Zero-context equivalent terms) *Given sets $L, R \subseteq \Sigma \times \Sigma$ of pairs of binary function symbols and constants, $\overset{L,R}{\cong}_0$ is the smallest equivalence relation satisfying the following constraints, for each $s \in \mathbb{T}(\Sigma)$:*

1. $\forall (f, c) \in L.\ c \overset{L,R}{\cong}_0 f(c, s)$, *and*
2. $\forall (g, d) \in R.\ d \overset{L,R}{\cong}_0 g(s, d)$.

We say that two terms $s, t \in \mathbb{T}(\Sigma)$ are zero-context equivalent, *if $s \overset{L,R}{\cong}_0 t$.*

Since the sets L and R are always clear from the context, in the remainder we write $\cong_0$ in place of $\overset{L,R}{\cong}_0$.

Theorem 3.3.1 (Decidability of zero-context equivalence) *Let $L, R \subseteq \Sigma \times \Sigma$ be finite sets of pairs of binary function symbols and constants. Then, for all terms $t, u \in \mathbb{T}(\Sigma)$, it is decidable whether $t \overset{L,R}{\cong}_0 u$ holds.*

Proof. Let L and R be given. Suppose the are given two terms t and u and we want to check whether they are zero-context equivalent. From t and u, construct the (undirected) graph $G(t, u)$ as follows.

The nodes in $G(t, u)$ are

- t and u,
- the constants mentioned in L and R,
- all terms of the form $f(c, d)$ with $(f, c) \in L$ and $(f, d) \in R$, and

The edges in $G(t, u)$ are given by items 1 and 2 in Definition 3.3.1. This graph is finite, since L and R are finite, and can be built effectively. Note that $G(u, t)$ and $G(t, u)$ are identical.

We claim that t is zero-context equivalent to u iff t can be reached from u in $G(t, u)$.

The proof of this claim is as follows. The right-to-left implication is immediate since each edge in $G(t, u)$ corresponds to an application of item 1 or item 2 in Definition 3.3.1. For the converse, we proceed by induction on the length of a shortest proof of $t \overset{L,R}{\cong}_0 u$. If $t \overset{L,R}{\cong}_0 u$ follows by reflexivity or by using item 1 or 2 in Definition 3.3.1 then t can be reached from u in $G(t, u)$ in zero steps or in one step, respectively. If $t \overset{L,R}{\cong}_0 u$ is proven using symmetry then the claim follows by the inductive hypothesis. Assume now that $t \overset{L,R}{\cong}_0 u$ follows by transitivity. Then there is some term s such that $t \overset{L,R}{\cong}_0 s$ (in one step) and $s \overset{L,R}{\cong}_0 u$. By induction and the symmetry of reachability, s is reachable from t in $G(t, s)$ and s is reachable from u in $G(s, u)$. To see that u is reachable from t in $G(t, u)$, we now observe that s can be taken to be

- a constant mentioned in L or R, if $t = f(c, t')$ for some $(f, c) \in L$ or $t = f(t', c)$ for some $(f, c) \in R$, or
- if t is a constant c, a term of one of the following forms for some constant d:
  - $f(c, d)$, where $(f, c) \in L$ and $(f, d) \in R$, or
  - $f(d, c)$, where $(f, c) \in R$ and $(f, d) \in L$.

Indeed, assume, by way of example, that $t = c$ and $s = f(c, t')$, where $(f, c) \in L$ and t' is not a constant d such that $(f, d) \in R$. Then the proof of $s \overset{L,R}{\cong}_0 u$ could only proceed in the next step by going back to $t = c$, contradicting our assumption that it was a shortest proof of $t \overset{L,R}{\cong}_0 u$.

It follows that both $G(t, s)$ and $G(s, u)$ are subgraphs of $G(t, u)$, and therefore t is reachable from u in $G(t, u)$, as claimed. ⊠

We now proceed to define the rule format for left and right zero elements, which is the first main contribution of this chapter. Before doing so, however, it may be useful to discuss some examples, which highlight two of the key design criteria in the definition to follow.

Example 3.3.2 *Assume that a is the only action. Let 0 be a constant with deduction rule*

$$\frac{}{0 \mapsto 0}$$

Furthermore consider the binary operators $\_[n\rangle\_$, for $n \geq 0$, with deduction rules

$$\frac{x \mapsto x'}{x[0\rangle y \mapsto x'} \qquad \frac{x \mapsto x'}{x[n+1\rangle y \mapsto x'[n\rangle y} \qquad \frac{x \overset{a}{\to} x'}{x[n\rangle y \overset{a}{\to} x'[n\rangle y} \qquad \frac{x \not\overset{a}{\mapsto} \quad x \overset{a}{\mapsto}}{x[n\rangle y \mapsto y}$$

Assuming that the transition relation $\mapsto$ denotes unit time steps, $p[n\rangle q$ denotes that q will start only when p has finished in at most n time units. In order to prove that 0 is a left zero element for the operator $\_[n\rangle\_$ one needs to show also that it is a left zero element for all operators $\_[i\rangle\_$ with $0 \leq i < n$. It is not hard to see that the relation

$$\mathcal{R}_n = \{(0[i\rangle p, 0) \mid 0 \leq i \leq n,\ p \in \mathbb{C}(\Sigma)\} \cup \{(0, 0)\}$$

is a bisimulation. Therefore, 0 is a left zero element for the operator $[n\rangle$. ⊠

In the previous example, the zero element property for $\_[n\rangle\_$ depends on that property for all $\_[i\rangle\_$ with $0 \leq i < n$. The next example illustrates that this dependency can even be worse.

Example 3.3.3 *Assume that a is the only action and consider the binary operators f_i, $i \geq 0$, with rules*

$$\frac{x_0 \overset{a}{\to} y_0}{f_i(x_0, x_1) \overset{a}{\to} f_{i+1}(y_0, x_1)}\ .$$

Let RUN_a be a constant with rule $RUN_a \xrightarrow{a} RUN_a$. Then $f_i(RUN_a, p) \leftrightarrow RUN_a$, for each closed term p and $i \geq 0$. Indeed, it is not hard to see that the relation

$$\mathcal{R} = \{(f_i(RUN_a, p), RUN_a) \mid i \geq 0,\ p \in \mathbb{C}(\Sigma)\}$$

is a bisimulation. Therefore, RUN_a is a left zero element for each of the operators f_i, $i \geq 0$. Note that, in order to show that RUN_a is a left zero element for, say, f_0, we need to consider a set of operators, namely $\{f_i \mid i \geq 0\}$. Moreover, such a set cannot be inductively defined since, in order to show that RUN_a is a left zero element for f_i, $i \geq 0$, we need to prove that RUN_a is a left zero element for f_{i+1}. Therefore the set of proof obligations is not well-founded. ⊠

Example 3.3.4 *Consider the following TSS with constant RUN_a and binary function symbols f and g with rules*

$$\frac{x_0 \xrightarrow{a} y_0}{f(x_0, x_1) \xrightarrow{a} g(x_1, y_0)} \qquad \frac{x_1 \xrightarrow{a} y_1}{g(x_0, x_1) \xrightarrow{a} f(y_1, x_0)}$$

It is not hard to see that $f(RUN_a, p) \leftrightarrow RUN_a \leftrightarrow g(p, RUN_a)$, for each closed term p. Therefore RUN_a is a left zero element for f and a right zero element for g. In the light of the mutual dependency between f and g, this example indicates that a widely applicable rule format for left zero elements will need to be based at the same time on a rule format for right zero elements, and vice versa. ⊠

In order to remain in line with the terminology in [22], in the following definition we talk about left- and right-aligned pairs.

Definition 3.3.5 (Left- and right-aligned pairs) *Given a TSS with set of predicate symbols $\mathcal{P}$ and set of labels $\mathcal{L}$, the sets L and R of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.*

1. *For each $(f, c) \in L$, the following conditions hold.*
   (a) *Whenever an axiom $c \xrightarrow{a} t$ (or $P\ c$) does exist then there is a rule:*

$$\frac{\{x_0 \xrightarrow{a_i} t_i \mid i \in I\} \cup \{P_k\ x_0 \mid k \in K\} \cup \{x_0 \xrightarrow{a_j} \text{ or } \neg P_j\ x_0 \mid j \in J\}}{f(x_0, x_1) \xrightarrow{a} t' \qquad (\text{or } P\ f(x_0, x_1))}$$

   where
   i. $x_1 \notin \{x_0\} \cup \bigcup_{i \in I} vars(t_i)$,

ii. *for each $j \in J$, there is no c-defining axiom with a_j or P_j as label (depending on whether the formula with index j is a transition or a predicate formula),*

iii. *there exists a collection $\{P_k\ c \mid k \in K\}$ of c-defining axioms, and*

iv. *there exists some substitution σ such that $\sigma(x_0) = c$, $\{c \xrightarrow{a_i} \sigma(t_i) \mid i \in I\}$ is included in the collection of c-defining axioms, and if the conclusion is a transition formula, $\sigma(t') \cong_0 t$.*

(b) *Each f-defining deduction rule has one of the following forms:*

$$\frac{\Phi}{f(t_0, t_1) \xrightarrow{a} t'} \quad \textit{or} \quad \frac{\Phi}{P\ f(t_0, t_1)}$$

where $a \in \mathcal{L}$, $P \in \mathcal{P}$ and, for each closed substitution σ such that $\sigma(t_0) \equiv c$, one of the following cases holds:

i. *there exists an axiom $c \xrightarrow{a} t$ with $\sigma(t') \cong_0 t$ (if the conclusion is a transition formula), or an axiom $P\ c$ (if the conclusion is a predicate formula), or*

ii. *there exists a premise $\phi \in \Phi$ with t_0 as its source such that*

A. *either ϕ is a positive formula and the collection of c-defining axioms does not entail $\sigma(\phi)$, or*

B. *ϕ is a negative formula and the collection of c-defining axioms contradicts $\sigma(\phi)$.*

2. *The definition of right-aligned pairs of operators and constant symbols—that is, those such that $(f, c) \in R$—is symmetric and is not repeated here.*

For a function symbol f and a constant c, we call (f, c) left aligned *(respectively,* right aligned*) if $(f, c) \in L$ (respectively, $(f, c) \in R$).*

The structure of the above definition is inherited directly from [22], but there are, however, significant differences in the details. Intuitively, the aim of condition 1a is to ensure that whenever the constant c performs, say, an a-transition then also $f(c, p)$ does so for each closed term p, and the two transitions lead to terms that are zero-context equivalent. Conversely, condition 1b guarantees that each transition that $f(c, p)$ can perform actually simulates one of the steps of the constant c. The clauses play the corresponding role also for predicates.

Note that, as in [22], the sets L and R are defined as the largest sets of pairs satisfying the constraints from Definition 3.3.5. This means that, in order to check

whether a constant c is, for example, a left zero element for an operator f, it is sufficient that the pair (f, c) be contained in L for a pair of sets L and R that satisfy the conditions above.

The following theorem states the correctness of the rule format in Definition 3.3.5.

Theorem 3.3.2 *Let $\mathcal{T}$ be a complete TSS in which each rule is f-defining for some function symbol f. Assume that L and R are the sets of left- and right-aligned function symbols according to Definition 3.3.5. For each $(f, c) \in L$, it holds that $f(c, x) \leftrightarrow c$. Symmetrically, for each $(f, c) \in R$, it holds that $f(x, c) \leftrightarrow c$.*

Proof. Section 3.9 contains the proof of Theorem 3.3.2. ⊠

Example 3.3.6 *Consider Example 3.3.3. We now show that RUN_a is a left zero element for each f_i using Theorem 3.3.2. To this end, let $L = \{(f_i, RUN_a) \mid i \geq 0\}$ and take $R = \emptyset$. Let us focus on a generic function symbol f_i. We prove that conditions 1a and 1b are met.*

- *1a: For the only axiom $RUN_a \xrightarrow{a} RUN_a$ we can use the only f_i-defining rule. Here we can associate the axiom $RUN_a \xrightarrow{a} RUN_a$ to the premise $x_0 \xrightarrow{a} y_0$ and consider a substitution σ such that $\sigma(x_0) \equiv \sigma(y_0) \equiv RUN_a$. Since $\sigma(y_0) \equiv RUN_a$ and $(f_{i+1}, RUN_a) \in L$, it follows that*

$$\sigma(f_{i+1}(y_0, x_1)) \equiv f_{i+1}(RUN_a, \sigma(x_1)) \cong_0 RUN_a \ ,$$

  and we are done.

- *1b: We can associate the only f_i-defining rule to the axiom $RUN_a \xrightarrow{a} RUN_a$. Assume that $\sigma(x_0) \equiv RUN_a$ but $\sigma(f_{i+1}(y_0, x_1)) \ncong_0 RUN_a$, and therefore case 1(b)i does not apply. This means that $\sigma(y_0) \not\equiv RUN_a$ and therefore the condition in case 1(b)iiA is met.* ⊠

Example 3.3.7 *Consider now Example 3.3.4. We show that RUN_a is a left zero element for f and a right zero element for g using Theorem 3.3.2. Let $L = \{(f, RUN_a)\}$ and $R = \{(g, RUN_a)\}$. We limit ourselves to checking that conditions 1a and 1b are met by the pair (f, RUN_a) contained in L.*

- *1a: For the only axiom $RUN_a \xrightarrow{a} RUN_a$, we can use the only rule for f. Indeed, the obvious substitution σ constructed as required in item 1(a)iv of Definition 3.3.5 satisfies that $\sigma(g(x_1, y_0)) \cong_0 RUN_a$ because $(g, RUN_a) \in R$.*

- *1b: The only f-defining rule is the one on the left. For that we can consider the axiom $RUN_a \xrightarrow{a} RUN_a$. If $\sigma(y_0) \equiv RUN_a$ then case 1(b)i applies since $(g, RUN_a) \in R$. Otherwise, the condition in case 1(b)iiA is met.*

A similar reasoning can be applied to the pair (g, RUN_a) *in* R. ⊠

We conclude this section by discussing some of the constraints in Definition 3.3.5 in order to argue that they cannot be easily relaxed. In what follows, we focus on the conditions that left-aligned pairs must meet. First of all, note that relaxing the requirement that $x_0 \not\equiv x_1$ in condition 1(a)i would jeopardize Theorem 3.3.2. To see this, consider the TSS with constant RUN_a and binary operator f with rule

$$\frac{x_0 \xrightarrow{a} y_0}{f(x_0, x_0) \xrightarrow{a} y_0} \ .$$

It is not hard to check that $L = \{(f, \text{RUN}_a)\}$ and $R = \emptyset$ satisfy all the constraints in Definition 3.3.5 apart from $x_0 \not\equiv x_1$. For example, let us examine condition 1b. Let σ be a closed substitution such that $\sigma(x_0) \equiv \text{RUN}_a$ and assume that the axiom for RUN_a entails $\sigma(x_0) \equiv \text{RUN}_a \xrightarrow{a} \sigma(y_0)$—or else condition 1(b)iiA would be met. It follows that $\sigma(y_0) \equiv \text{RUN}_a$ and therefore condition 1(b)i is satisfied.

However, RUN_a is *not* a left zero element for f. For example, the term $f(\text{RUN}_a, f(\text{RUN}_a))$ affords no transition and therefore cannot be bisimilar to RUN_a.

The following example shows that relaxing the requirement that

$$x_1 \notin \bigcup_{i \in I} vars(t_i)$$

in condition 1(a)i would also invalidate Theorem 3.3.2. To see this, consider the TSS with constant RUN_a and binary operator f with rule

$$\frac{x_0 \xrightarrow{a} x_1}{f(x_0, x_1) \xrightarrow{a} x_1} \ .$$

Again, it is not hard to check that $L = \{(f, \text{RUN}_a)\}$ and $R = \emptyset$ satisfy all the constraints in Definition 3.3.5 apart from the requirement that x_1 should not occur in the target of a positive premise. However, $f(\text{RUN}_a, f(\text{RUN}_a, \text{RUN}_a))$ affords no transition and therefore cannot be bisimilar to RUN_a. This means that RUN_a is *not* a left zero element for f.

The role played by requirements 1(a)ii and 1(a)iv in ensuring that, modulo bisimilarity, $f(c, p)$ affords 'the same transitions as c', for each p, is highlighted by the following two examples.

Example 3.3.8 *Consider the TSS with constants* $\mathbf{0}$ *and* $a\&b$*, and a binary operator* f *with rules:*

$$\frac{}{a\&b \xrightarrow{a} \mathbf{0}} \qquad \frac{}{a\&b \xrightarrow{b} \mathbf{0}} \qquad \frac{x_0 \overset{b}{\not\to} \quad x_0 \xrightarrow{a} y_0}{f(x_0, x_1) \xrightarrow{a} y_0} \qquad \frac{x_0 \overset{a}{\not\to} \quad x_0 \xrightarrow{b} y_0}{f(x_0, x_1) \xrightarrow{b} y_0} \ .$$

It is not hard to check that $L = \{(f, a\&b)\}$ *and* $R = \emptyset$ *satisfy all the constraints in Definition 3.3.5 apart from 1(a)ii. However, the term* $f(a\&b, \mathbf{0})$ *affords no transition unlike* $a\&b$*. Therefore* $a\&b$ *is not a left zero element for* f*.* ⊠

Example 3.3.9 *Consider the TSS over set of labels* $\{a, b\}$ *with constant* RUN_a *and a binary operator* f *with rule:*

$$\frac{x_0 \xrightarrow{a} y_0 \quad x_0 \xrightarrow{b} y_1}{f(x_0, x_1) \xrightarrow{a} x_1} \ .$$

It is easy to check that $L = \{(f, RUN_a)\}$ *satisfies all the constraints in Definition 3.3.5 apart from 1(a)iv. However,* $f(RUN_a, RUN_a)$ *affords no transition unlike* RUN_a*. Therefore* RUN_a *is* not *a left zero element for* f*.* ⊠

As witnessed, e.g., by Example 3.4.4 to follow, constraint 1(b)i enhances the generality of our format. Indeed, if we removed constraint 1(b)i and a left-aligned pair (f, c) satisfied condition 1(b)ii, then no rule for f would be applicable to a closed term of the form $f(c, p)$. Therefore, no term of the form $f(c, p)$ would afford a transition. Since (f, c) satisfies condition 1 in Definition 3.3.5, the collection of c-defining axioms must be empty. As a consequence, the resulting format would be unable to handle left zero elements such as RUN_a that afford some transition. Examples of constants with deduction axioms in the literature are immediate deadlock [29], which acts as a left zero element for sequential composition, parallel composition, left merge and communication merge and as a right zero element for parallel composition and communication merge, and delayable deadlock from [24], which is a left zero element for sequential composition.

3.4 Examples

In this section we show that several examples of zero elements from the literature indeed fit the format described in Section 3.3.

Example 3.4.1 (Synchronous parallel composition) *Consider the synchronous parallel composition from CSP [77] over a set of actions* $\mathcal{L}$ *with rules:*

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_{\mathcal{L}} y \xrightarrow{a} x' \parallel_{\mathcal{L}} y'} \quad (a \in \mathcal{L}) \ .$$

We know that the inaction constant $\mathbf{0}$*, with no rules, is a left and right zero element for* $\parallel_{\mathcal{L}}$*. Let* $L = R = \{(\parallel_{\mathcal{L}}, \mathbf{0})\}$*. We claim that* L *and* R *meet the constraints in Definition 3.3.5. First of all,* $\mathbf{0}$ *has no axioms so the clauses 1a and its symmetric counterpart 2a are vacuously satisfied. To show that also the clause 1b is met, we consider the rule above and note that, for every possible substitution* σ *such that* $\sigma(x) \equiv \mathbf{0}$*, the empty set of deduction rules does not entail the premise* $\sigma(x) \xrightarrow{a} \sigma(x')$*. This meets constraint 1(b)iiA. The symmetric counterpart of clause 1b is handled in similar fashion. The well-known laws*

$$\mathbf{0} \parallel_{\mathcal{L}} y \leftrightarrows \mathbf{0} \quad \textit{and} \quad x \parallel_{\mathcal{L}} \mathbf{0} \leftrightarrows \mathbf{0}$$

thus follow from Theorem 3.3.2. ⊠

Example 3.4.2 (Left merge operator) *Consider the left merge operator from [37].*

$$\frac{x \xrightarrow{a} x'}{x \mathbin{\underline{\parallel}} y \xrightarrow{a} x' \parallel y}$$

Here $\parallel$ *stands for the merge operator from [37], whose SOS specification is immaterial for this example; see Example 3.4.6 to follow. Let* $L = \{(\mathbin{\underline{\parallel}}, \mathbf{0})\}$ *and* $R = \emptyset$*. We claim that* L *meets the constraints in Definition 3.3.5. It is easy to check that the claim is true by the same reasoning used in Example 3.4.1. This time it is sufficient to check conditions 1a and 1b because* $\mathbf{0}$ *is just a* left *zero element for* $\mathbin{\underline{\parallel}}$*. By Theorem 3.3.2 the validity of the law* $(\mathbf{0} \mathbin{\underline{\parallel}} y) \leftrightarrows \mathbf{0}$ *follows. Note that the pair* $\{(\mathbin{\underline{\parallel}}, \mathbf{0})\}$ *cannot be added to* R *because the symmetric version of condition 1b would be violated. Indeed* $\mathbf{0}$ *is* not *a right zero element for* $\mathbin{\underline{\parallel}}$*.* ⊠

Example 3.4.3 (Sequential Composition (1)) *We now examine an example that involves the use of predicates. Consider the standard sequential composition operator* $\cdot$*, which makes use of the predicate symbol* $\downarrow$*. (The formula* $x \downarrow$ *means that* x *terminates successfully.)*

$$(seq1)\ \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad (seq2)\ \frac{x \downarrow \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'} \qquad (seq3)\ \frac{x \downarrow \quad y \downarrow}{(x \cdot y) \downarrow}$$

Consider the deadlock constant δ*, defined by no axioms. In particular,* $\delta \downarrow$ *does* not *hold.*

Let $L = \{(\cdot, \delta)\}$ *and* $R = \emptyset$*. We claim that* L *meets the constraints in Definition 3.3.5. Here again condition 1a is vacuously true. In order to show that constraint 1b is also*

satisfied, consider a substitution σ that maps x to δ. It suffices only to observe that each of the above rules has a positive premise ϕ such that $\sigma(\phi)$ is not entailed by the empty set of rules. Therefore, once again, we fall under case 1(b)iiA. By Theorem 3.3.2, the validity of the well-known law $\delta \cdot y \leftrightarrows \delta$ follows.

Note that the pair $\{(\cdot, \delta)\}$ cannot be added to R because rule (seq1) would invalidate the symmetric counterpart of condition 1b in Definition 3.3.5. Indeed δ is not *a right zero element for $\cdot$.* ⊠

Example 3.4.4 (Sequential Composition (2)) *Focusing again on the sequential composition operator from the previous example, consider once more the constant RUN_a from Example 3.3.3 with axiom*

$$\frac{}{RUN_a \xrightarrow{a} RUN_a} \ .$$

This constant simply displays a infinitely many times. This behaviour is enough to preempt the execution of the right-hand argument of $\cdot$ and our order of business in this example is indeed to check the validity of the laws $RUN_a \cdot y \leftrightarrows RUN_a$ with $a \in \mathcal{L}$ using Theorem 3.3.2.

Let $L = \{(\cdot, RUN_a)\}$ and $R = \emptyset$. We claim that L meets the constraints in Definition 3.3.5. To prove this claim, we consider each constraint in turn.

- *1a: We need to match the above axiom for RUN_a with a rule that defines $\cdot$. The rule we pick is the instance of (seq1) for action a. The substitution σ constructed in order to meet the requirements in condition 1(a)iv is such that $\sigma(x) \equiv RUN_a$ and $\sigma(x') \equiv RUN_a$. Moreover, RUN_a is zero-context equivalent to $RUN_a \cdot y$ and we are done.*
- *1b: Since $RUN_a \downarrow$ does not hold, with the rules (seq2) and (seq3) we fall in the subcase 1(b)iiA. The rule (seq1) falls instead in the subcase 1(b)i for the same reason of the case 1a examined above.*

Note that, following the above reasoning, we can show the validity of laws of the form $c \cdot y \leftrightarrows c$, where c is any constant whose behaviour is defined by a collection of axioms of the form $\{c \xrightarrow{a_i} c \mid i \in I\}$, where I is any index set. ⊠

Example 3.4.5 (Predictable failure constant of $BPA_{0\delta}$) *In this example we focus on the language $BPA_{0\delta}$ of Baeten and Bergstra—see [25]. The* predictable failure 0 *is a non-standard constant that 'absorbs the computation' no matter where it appears within the context of the sequential composition operator $\cdot$. Namely, the laws $x \cdot 0 \leftrightarrows 0$ and $0 \cdot x \leftrightarrows 0$ both hold. The following SOS rules for the language $BPA_{0\delta}$ make use of the*

predicate $\neq 0$ *that determines whether or not a process can be proved equal to* 0*, and of predicates* $\xrightarrow{a} \checkmark$ *that tell us when a process can terminate by performing an a action.*

$$\frac{}{a \neq 0} \qquad \frac{}{\delta \neq 0} \qquad \frac{}{a \xrightarrow{a} \checkmark} \qquad \frac{x \neq 0}{x + y \neq 0} \qquad \frac{y \neq 0}{x + y \neq 0}$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} \checkmark}{x + y \xrightarrow{a} \checkmark} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{y \xrightarrow{a} \checkmark}{x + y \xrightarrow{a} \checkmark}$$

$$\frac{x \neq 0 \quad y \neq 0}{x \cdot y \neq 0} \qquad \frac{x \xrightarrow{a} x' \quad y \neq 0}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x \xrightarrow{a} \checkmark \qquad y \neq 0}{x \cdot y \xrightarrow{a} y}$$

Let $L = R = \{(\cdot, 0)\}$*. We claim that* L *and* R *meet the constraints in Definition 3.3.5. Firstly,* 0 *has no axioms so the clause 1a and its symmetric counterpart are vacuously satisfied. To show that clause 1b is satisfied, we must consider the three rules for* $\cdot$ *one by one. Since* $0 \neq 0$ *does not hold we fall into case 1(b)ii with the leftmost rule. Since* $0 \not\xrightarrow{a}$ *and* $0 \not\xrightarrow{a} \checkmark$ *for any* a*, the remaining rules also fall into the case 1(b)iiA. The symmetric counterpart of condition 1b is satisfied for each of the rules because* $0 \neq 0$ *does not hold. The laws*

$$x \cdot 0 \leftrightarrows 0 \quad \textit{and} \quad 0 \cdot x \leftrightarrows 0$$

thus follow by Theorem 3.3.2. ⊠

Example 3.4.6 (Merge operator) *Let* $\mathcal{L}$ *be the set of actions. Consider the classic merge operator* $\|$ *with the following rules, where* $a \in \mathcal{L}$*.*

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

Let $RUN_{\mathcal{L}}$ *be a constant defined by axioms* $RUN_{\mathcal{L}} \xrightarrow{a} RUN_{\mathcal{L}}$ *for each action* $a \in \mathcal{L}$*. We claim that the constant* $RUN_{\mathcal{L}}$ *is both a left and right zero element for* $\|$*. This can be checked using Theorem 3.3.2. Indeed, let* $L = R = \{(\|, RUN_{\mathcal{L}})\}$*. It is easy to see that condition 1a in Definition 3.3.5 is met for* $RUN_{\mathcal{L}} \xrightarrow{a} RUN_{\mathcal{L}}$ *by taking the instance of the left-hand rule for* $\|$ *with action* a*. Moreover, such a rule also meets condition 1(b)i.*

Consider now the right-hand rule for $\|$ *with action* a*. That rule also meets condition 1(b)i. Indeed, for each closed substitution* σ *such that* $\sigma(x) \equiv RUN_{\mathcal{L}}$*, we have that*

$$\sigma(x \parallel y') \equiv RUN_{\mathcal{L}} \parallel \sigma(y') \cong_0 RUN_{\mathcal{L}}$$

and $RUN_{\mathcal{L}} \xrightarrow{a} RUN_{\mathcal{L}}$ *is one of the axioms for the constant* $RUN_{\mathcal{L}}$*.* ⊠

Example 3.4.7 (A right-choice operator) *In this example we apply our format to a non-standard operator. For the sake of simplicity we assume that a is the only action. Consider a variant of the choice operator of Milner's CCS [96], where the right-hand argument has a higher priority than the left-hand argument, i.e., the scheduler executes the left-hand argument only when the other one has no transitions. The rules for such an operator are as follows:*

$$\frac{x \xrightarrow{a} x' \quad y \overset{a}{\not\to}}{x \mathrel{+\!\!\!\leftarrow} y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x \mathrel{+\!\!\!\leftarrow} y \xrightarrow{a} y'} \ .$$

Let c be any constant whose behaviour is defined by a non-empty, finite collection of axioms $\{c \xrightarrow{a} p_i \mid i \in I\}$*, where I is some index set. Reasoning as in the previous examples, using Theorem 3.3.2, we are able to prove the validity of the law* $x \mathrel{+\!\!\!\leftarrow} c \leftrightarrows c$*. We leave the details to the reader. The operator studied in this example bears resemblance with the preferential choice operator* $\twoheadrightarrow$ *from [39].* ⊠

3.5 Discussion of the format

The format for left and right zero elements we presented in Definition 3.3.5 is based on rather intuitive design decisions and, as witnessed by the examples discussed in Section 3.4, it is applicable to several operators from the literature. However, the format does have some, mostly theoretical, limitations and can be modified in several ways in order to improve some of its features. After all, the design of rule formats for SOS is always based on a trade-off between generality and applicability, and is, to some extent, an 'experimental science'.

Below we discuss two features of the rule format described in Definition 3.3.5. This discussion will motivate the development of an alternative format for left and right zero elements that we present in Section 3.6 to follow.

3.5.1 Premises of rules

One of the main potential limitations of the format for left zero elements is that the form of the rules in condition 1a does *not* allow one to test the variable x_1 in the premises; that is, we are able to test only the variable that will be instantiated with the left zero element c. The reason for this design choice is as follows. When an axiom $c \xrightarrow{a} t$ is present, we must ensure that, regardless of the second argument

of f, at least one rule for f proving an a-labelled transition does fire (leading to a term that is bisimilar to t). The way we guarantee this property stems from [22], i.e., we judiciously manage the presence/absence of c-defining axioms in order to satisfy the premises. Moreover, we require a strong syntactic connection between the closed term that is the target of the axiom $c \xrightarrow{a} t$ and the instantiated target of the conclusion of the rule for f. The same reasoning underlies our design choices for c-defining axioms of the form $P\,c$, where P is a predicate symbol.

In condition 1(b)ii, we must ensure that the rule under consideration either *cannot* fire when the first argument of f is instantiated with c or otherwise it would lead to a term that is bisimilar to a derivative of the left zero element.

In both of the aforementioned situations, it is important to reason about the satisfiability of premises of rules. The conditions we give in 1a and 1(b)ii can be indeed regarded as a basic, syntactic approximation of our attempt to ensure firabily/unfirability of the rules in question, when the first argument of the operator f is the considered left zero element. Premises about the argument x_1 are a challenge, because, since x_1 can be replaced by an arbitrary closed term, there is no easy, purely syntactic way to ensure their satisfiability in the context of a left zero element c. For this reason, testing x_1 is forbidden by the format for left zero elements in Definition 3.3.5. However, this choice does not allow us to handle left zero elements such as the one in the following example.

Example 3.5.1 *Consider a TSS with constants* $\mathbf{0}$ *and* RUN_a *(from Example 3.3.3), and a function symbol* f *defined as follows*

$$(y)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{f(x,y) \xrightarrow{a} x'} \qquad (not\check{}y)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} \not\to}{f(x,y) \xrightarrow{a} x'}\ .$$

The constant RUN_a *is a left zero element for* f*, but the pair* (f, RUN_a) *is not left aligned because the test on the variable* y *is forbidden by condition 1a in Definition 3.3.5.* ⊠

This example is admittedly highly artificial. (Indeed, we are not aware of any operator from the literature that is specified using rules akin to the ones given above.) The following one is perhaps less so.

Example 3.5.2 *Assume that a is the only action. Consider the TSS with constant RUN_a from Example 3.3.3 and binary operator f with rule*

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{f(x,y) \xrightarrow{a} f(x',y')} \ .$$

We claim that the constant RUN_a is a left and a right zero element for f. Indeed, each closed term in the TSS above is bisimilar to RUN_a. On the other hand, the pair (f, RUN_a) is neither left- nor right-aligned because of the premises involving y and x in the rule for f, respectively. ⊠

Admittedly, neither of the examples given above is to be found in the literature. However, we feel that the study of versions of our rule format that allow one to test both arguments of a binary operator is a natural question to address. In Section 3.6.2, we propose a format, based on the GSOS format of Bloom, Istrail and Meyer, that is able to handle the examples we mentioned above and that has independent interest.

3.5.2 Checking the format, algorithmically

We are aware that checking the requirements in Definition 3.3.5 may involve hard work. Namely, they require the user to provide proofs of zero-context equivalence between terms and of entailment/contradiction between collections of transition formulae. This is not an unexpected drawback because it is inherited from the design of the format for left and right unit elements from [22].

Even though the requirements of the proposed format are frequently easy to check in practice, as the examples in Section 3.4 clearly indicate, their verification may be very lengthy in general and steps toward alternative mechanizable solutions are desirable.

In the next section, our order of business is to provide an alternative rule format for zero elements, which is a candidate for automated checking and retains enough expressiveness to cover relevant examples from the literature, such as those we presented in Section 3.4.

3.6 A rule format for zero elements based on GSOS

In what follows, we adapt the format from Section 3.3 in the context of GSOS languages. By employing the *logic of initial transitions* developed in [7], we are able to reason easily about firability of rules, and the resulting rule format is a step towards addressing both the issues discussed in Sections 3.5.1 and 3.5.2.

3.6.1 The logic of initial transitions

In this section, for the sake of completeness, we discuss the logic we employ in the definition of our revised rule format for left and right zero elements based on GSOS. The *logic of initial transitions* has been recently introduced in [7] in order to reason about the satisfiability of the premises of GSOS rules. The set of initial transitions formulae over a finite set of actions $\mathcal{L}$ is defined by the following grammar, where $a \in \mathcal{L}$:

$$F ::= \quad \mathsf{True} \mid x \xrightarrow{a} \mid \neg F \mid F \wedge F \ .$$

As usual, we write False for $\neg\mathsf{True}$, and $F \vee F'$ for $\neg(\neg F \wedge \neg F')$.

The semantics of this logic is given by a satisfaction relation $\models$ that is defined, relative to a GSOS language $G = (\Sigma_G, \mathcal{L}, R_G)$, by structural recursion on F in the following way, where σ is a closed substitution and $\rightarrow_G$ is the collection of transitions that can be proven using the rules in R_G:

$$\begin{array}{rcl}
\rightarrow_G, \sigma \models \mathsf{True} & & \text{always} \\
\rightarrow_G, \sigma \models x \xrightarrow{a} & \Leftrightarrow & \sigma(x) \xrightarrow{a}_G p, \text{ for some } p \\
\rightarrow_G, \sigma \models \neg F & \Leftrightarrow & \text{not } \rightarrow_G, \sigma \models F \\
\rightarrow_G, \sigma \models F \wedge F' & \Leftrightarrow & \rightarrow_G, \sigma \models F \text{ and } \rightarrow_G, \sigma \models F' \ .
\end{array}$$

The reader familiar with Hennessy-Milner logic [74] will have noticed that the propositions of the form $x \xrightarrow{a}$ correspond to Hennessy-Milner formulae of the form $\langle a \rangle \mathsf{True}$. In what follows, we consider formulae up to commutativity and associativity of $\wedge$.

We use the logic to turn the set of premises Φ of a GSOS rule into a formula that describes the collection of closed substitutions that satisfy Φ.The conversion

procedure hyps is borrowed from [7]. Formally,

$$\begin{aligned} \mathsf{hyps}(\emptyset) &= \mathsf{True} \\ \mathsf{hyps}(\{x \xrightarrow{a} y\} \cup \Phi) &= (x \xrightarrow{a}) \wedge \mathsf{hyps}(\Phi \setminus \{x \xrightarrow{a} y\}) \\ \mathsf{hyps}(\{x \overset{a}{\nrightarrow}\} \cup \Phi) &= \neg(x \xrightarrow{a}) \wedge \mathsf{hyps}(\Phi \setminus \{x \overset{a}{\nrightarrow}\}) \ . \end{aligned}$$

Intuitively, if Φ is the set of premises of a rule then $\mathsf{hyps}(\Phi)$ is the conjunction of the corresponding initial transition formulae. For example,

$$\mathsf{hyps}(\{x \xrightarrow{a} y, z \overset{b}{\nrightarrow}\}) = (x \xrightarrow{a}) \wedge \neg(z \xrightarrow{b}) \ .$$

If J is a finite set of GSOS rules, we overload hyps and write:

$$\mathsf{hyps}(J) = \bigvee_{r \in J} \mathsf{hyps}(\Phi_r) \ ,$$

where Φ_r is the set of premises of rule r.

Lemma 3.6.1 *Assume that G is a GSOS language. Let $\Phi = \Phi_1 \cup \Phi_2$, where Φ_1 and Φ_2 are disjoint, be the set of premises of a rule in G of the form (3.1) on page 72. Let σ be a closed substitution such that $\rightarrow_G, \sigma \models \mathsf{hyps}(\Phi)$ and σ satisfies Φ_1. Then there is a closed substitution σ' such that*

- *$\sigma'(x_i) = \sigma(x_i)$ for each $i \in \{1, \ldots, l\}$,*
- *$\sigma'(y) = \sigma(y)$ for each target variable y of a positive premise in Φ_1 and*
- *σ' satisfies Φ.*

Proof. We construct a substitution σ' meeting the requirements stated in the lemma by induction on the cardinality of Φ_2. If Φ_2 is empty, then take σ' to be σ. Otherwise, pick an arbitrary transition formula in Φ_2. If the transition formula is of the form $x_i \overset{b}{\nrightarrow}$, for some $i \in \{1, \ldots, l\}$ and label b, then $\neg(x_i \xrightarrow{b})$ is a conjunct of $\mathsf{hyps}(\Phi)$. As $\rightarrow_G, \sigma \models \mathsf{hyps}(\Phi)$, we have that σ satisfies $x_i \overset{b}{\nrightarrow}$. Therefore σ satisfies $\Phi_1 \cup \{x_i \overset{b}{\nrightarrow}\}$ and the existence of a substitution σ' meeting the requirements stated in the lemma follows by induction applied to $\Phi_2 \setminus \{x_i \overset{b}{\nrightarrow}\}$.

Consider now the case that $x_i \xrightarrow{a} y \in \Phi_2$ for some variable y and label a. As $\rightarrow_G, \sigma \models \mathsf{hyps}(\Phi)$ and $x_i \xrightarrow{a}$ is a conjunct of $\mathsf{hyps}(\Phi)$, we have that $\sigma(x_i) \xrightarrow{a} p$ for some closed term p. Let σ'' be the closed substitution that maps the variable y to p and agrees with σ on all the other variables. Since all the variables in a GSOS rule are distinct, and Φ_1 and Φ_2 are disjoint, σ'' satisfies $\Phi_1 \cup \{x_i \xrightarrow{a} y\}$.

Moreover, by construction, σ and σ'' agree on the variables occurring in the source of the conclusion of the rule and on each target variable y' of a premise in Φ_1. The existence of a substitution σ' meeting the requirements stated in the lemma follows now by induction applied to $\Phi_2 \setminus \{x_i \xrightarrow{a} y\}$. ⊠ We write $\models_G F \Rightarrow F'$ iff every substitution that satisfies F also satisfies F'. This semantic entailment preorder is decidable, as shown in [7].

Theorem 3.6.1 (Decidability of entailment [7]) *Let G be a GSOS language. Then, for all formulae F and F', it is decidable whether $\models_G F \Rightarrow F'$ holds.*

As a matter of fact, when Φ is the set of the premises of a rule r, checking whether $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(\Phi)$ holds is equivalent to checking whether the rule r is always firable. Conversely, checking whether $\models_G \mathsf{hyps}(\Phi) \Rightarrow \mathsf{False}$ holds is equivalent to checking whether the rule r never fires. These considerations will be useful in the remainder of the chapter. Our definition of the alternative rule format for left and right zero elements makes use of the logic and especially of these two kinds of entailment. The semantic entailment is, moreover, used in a simplified fashion where one does not need to check all the closed substitutions, but only those that map one variable to the left or right zero element constant under consideration. We now proceed to formalize this notion.

Definition 3.6.2 *Let $G = (\Sigma_G, \mathcal{L}, R_G)$ be a GSOS language. For each formula F, constant $c \in \Sigma_G$ and variable x, we define the formula $F[x \mapsto c]$ by structural recursion on F as follows:*

$$
\begin{aligned}
\mathit{True}[x \mapsto c] &= \mathit{True} \\
(x \xrightarrow{a})[x \mapsto c] &= \begin{cases} \mathit{True} & \text{if there is a } c\text{-defining axiom } c \xrightarrow{a} p \text{ for some } p \\ \mathit{False} & \text{otherwise} \end{cases} \\
(y \xrightarrow{a})[x \mapsto c] &= y \xrightarrow{a} \quad \text{if } x \neq y \\
(\neg F)[x \mapsto c] &= \neg(F[x \mapsto c]) \\
(F_1 \wedge F_2)[x \mapsto c] &= (F_1[x \mapsto c]) \wedge (F_2[x \mapsto c]) \ .
\end{aligned}
$$

The connection between F and $F[x \mapsto c]$ is provided by the following lemma.

Lemma 3.6.3 *Let $G = (\Sigma_G, \mathcal{L}, R_G)$ be a GSOS language. Let F be a formula, c be a constant in Σ_G and x be a variable. Then, for each closed substitution σ,*

$$\rightarrow_G, \sigma \models F[x \mapsto c] \;\; \textit{iff} \;\; \rightarrow_G, \sigma[x \mapsto c] \models F \;\; ,$$

where $\sigma[x \mapsto c]$ denotes the substitution that maps x to c and acts like σ on all the other variables.

As a consequence of the above lemma, checking whether F holds for all substitutions that map variable x to a constant c amounts to showing that the formula $F[x \mapsto c]$ is satisfied by all substitutions—that is, showing that $F[x \mapsto c]$ is a tautology over G.

3.6.2 An alternative rule format for zero elements

We now have all the ingredients to reformulate the format we presented in Section 3.3 within the GSOS format. This time the conditions of our format will not try to ensure firability/unfirability of rules by purely syntactic means as in Definition 3.3.5, but they instead exploit the logic of initial transition formulae to incorporate a modicum of semantic reasoning within the rule format.

In what follows the reader should bear in mind that, by the considerations in Section 3.6.1 and by the disjunctive nature of $\mathsf{hyps}(J)$, with J set of rules, the semantic entailment $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(J)$ actually holds whenever, no matter what closed substitution σ we pick, at least one of the rules in the set J does fire, when it is instantiated with σ.

Definition 3.6.4 (Left- and right-aligned pairs, anew) *Let G be a GSOS language. The sets L and R of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.*

1. *For each $(f, c) \in L$, the following conditions hold.*
   (a) *For each axiom $c \xrightarrow{a} t$, there exists a set J of rules of the form*

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

   such that
   i. $\models_G \mathit{True} \Rightarrow \mathsf{hyps}(J)[x_0 \mapsto c]$, *and*
   ii. *for each rule in J, one of the following cases holds:*

A. *there is some variable* $y \in vars(t')$ *such that* $x_0 \xrightarrow{a} y \in \Phi$ *and* $\sigma(t') \cong_0 t$, *where* σ *is the substitution mapping* x_0 *to* c, y *to* t *and is the identity on all the other variables, or*

B. $\sigma(t') \cong_0 t$, *where* σ *is the substitution mapping* x_0 *to* c *and is the identity on all the other variables.*

(b) *For each* f*-defining deduction rule*

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

one of the following cases holds:

i. *there exists an axiom* $c \xrightarrow{a} t$ *such that*

A. *there is some variable* $y \in vars(t')$ *such that* $x_0 \xrightarrow{a} y \in \Phi$ *and* $\sigma(t') \cong_0 t$, *where* σ *is the substitution mapping* x_0 *to* c, y *to* t *and is the identity on all the other variables, or*

B. $\sigma(t') \cong_0 t$, *where* σ *is the substitution mapping* x_0 *to* c *and is the identity on all the other variables.*

ii. $\models_G \mathsf{hyps}(\Phi)[x_0 \mapsto c] \Rightarrow \mathsf{False}$.

2. *The definition of right-aligned pairs of operators and constant symbols—that is, those such that* $(f, c) \in R$*—is symmetric and is not repeated here.*

For a function symbol f *and a constant* c, *we call* (f, c) left aligned *(respectively,* right aligned*) if* $(f, c) \in L$ *(respectively,* $(f, c) \in R$*).*

Remark 3.6.5 *Let* G *be a GSOS language over a signature including at least one constant. Since* $\mathsf{hyps}(J)$ *is a disjunctive formula, condition 1(a)i in the above definition implies that the set* J *is non-empty. On the other hand, condition 1(b)ii says that the premises of the rule under consideration cannot be satisfied by any closed substitution that maps the variable* x_0 *to the constant* c.

In condition 1a and its symmetric counterpart, one must identify a set J *of rules. To understand why, the reader should consider Example 3.5.1, where the rules* (y) *and* $(not\text{–}y)$ *only together allow the operator* f *to simulate the behaviour of the constant* RUN_a*: no matter what closed term is substituted for the argument variable* y, *we are sure that one of the two rules fires and that the transition leads to* RUN_a. *In Definition 3.6.4, these two properties are guaranteed, respectively, by conditions 1(a)i and 1(a)ii.* ⊠

Theorem 3.6.2 *Let G be a GSOS language. Assume that L and R are the sets of left- and right-aligned function symbols according to Definition 3.6.4. For each $(f, c) \in L$, it holds that $f(c, x) \leftrightarrows c$. Symmetrically, for each $(f, c) \in R$, it holds that $f(x, c) \leftrightarrows c$.*

Section 3.10 contains the proof of Theorem 3.6.2.

The following result is a consequence of Theorems 3.3.1 and 3.6.1.

Theorem 3.6.3 *For GSOS languages, the sets L and R can be effectively constructed.*

Remark 3.6.6 (Handling predicates using the format of Definition 3.6.4) *The formats in Definition 3.3.5 and the one in Definition 3.6.4 are incomparable. Indeed the former allows for complex terms in the source of the conclusions of rules and in premises of rules. In addition, the format from Definition 3.3.5 does not require all variables in the premises of rules to be distinct and permits the use of predicates. GSOS languages forbid all of these features. On the other hand, it is easy to see that, using the format from Definition 3.6.4, one can check Example 3.5.1, which cannot be dealt with by the format from Definition 3.3.5.*

It is important to note, however, that the GSOS-based format we presented in Definition 3.6.4 can indeed be used to reason about the examples from Section 3.4 that use predicates. In fact, one can encode a finite collection of predicates specified using rules of the form

$$\frac{\{x_0 \xrightarrow{a_i} y_i \mid i \in I\} \cup \{P_k\, x_0 \mid k \in K\} \cup \{x_0 \overset{a_j}{\nrightarrow} \text{ or } \neg P_j\, x_0 \mid j \in J\}}{P\, f(x_0, x_1)}\ ,$$

where

- *the index sets I, K and J are finite and*
- *the variables x_0, x_1 and y_i, $i \in I$, are pairwise different,*

rather easily by means of transition relations specified by GSOS rules. One can find a number of such encodings in the literature—see, for instance, [49, 151]. The key idea in each of these encodings is that a predicate P is represented as a transition relation $\xrightarrow{P}$ (assuming that P is a fresh action label) with some fixed fresh constant c_P as target and a fresh variable for the target of each of the premises.

For example, using this encoding strategy, the above rule becomes the standard GSOS rule

$$\frac{\{x_0 \xrightarrow{a_i} y_i \mid i \in I\} \cup \{x_0 \xrightarrow{P_k} z_k \mid k \in K\} \cup \{x_0 \overset{a_j}{\nrightarrow} \text{ or } x_0 \overset{P_j}{\nrightarrow} \mid j \in J\}}{f(x_0, x_1) \xrightarrow{P} c_P}\ ,$$

where the variables z_k are fresh and pairwise distinct.

With this encoding of predicates, which preserves finiteness of a language specification, the format proposed in Definition 3.6.4 is immediately applicable to all the examples we discussed in Section 3.4, as well as to those mentioned in, e.g., [32]. ⊠

Example 3.6.7 *Consider again the TSS discussed in Example 3.5.2. We will now argue that the format proposed in Definition 3.6.4 is capable of proving the validity of the laws*

$$f(x, RUN_a) \leftrightarrow RUN_a \text{ and } f(RUN_a, y) \leftrightarrow RUN_a,$$

unlike the purely syntactic one we introduced in Section 3.3. To see this, take $L = R = \{(f, RUN_a)\}$. We limit ourselves to checking that conditions 1a and 1b in Definition 3.6.4 are met.

- *1a : The only axiom for RUN_a is $RUN_a \xrightarrow{a} RUN_a$. Take J as the set containing the single rule for f. Then*

$$(x \xrightarrow{a} \wedge\, y \xrightarrow{a})[x \mapsto RUN_a] = \mathit{True} \wedge y \xrightarrow{a} \quad .$$

  As we observed in Example 3.5.2, each closed term in the TSS under consideration affords an a-labelled transition. Therefore, the formula $\mathit{True} \wedge y \xrightarrow{a}$ is a tautology and condition 1(a)i is met. Note, moreover, that $x \xrightarrow{a} x'$ is a premise of the only rule for f, $x \in vars(f(x', y'))$ and $f(RUN_a, y') \cong_0 RUN_a$. Therefore condition 1(a)ii is also met.

- *1b : Reasoning as above, we can easily check that the only rule for f meets condition 1(b)iA.* ⊠

Example 3.6.8 *Consider now the synchronous parallel composition of Example 3.4.1. We claim that the format proposed in Definition 3.6.4 is capable of proving the validity of the laws $(\mathbf{0} \parallel_{\mathcal{L}} y) \leftrightarrow \mathbf{0}$ and $(x \parallel_{\mathcal{L}} \mathbf{0}) \leftrightarrow \mathbf{0}$.*

Let $L = R = \{(\parallel_{\mathcal{L}}, \mathbf{0})\}$. Since the constant $\mathbf{0}$ has no axioms, condition 1a is vacuously satisfied. In order to see that also condition 1b is satisfied, it is sufficient to notice that the only rule for $\parallel_{\mathcal{L}}$ can never fire because $\mathbf{0}$ has no transitions. Indeed, the entailment $\models_G (x \xrightarrow{a} \wedge\, y \xrightarrow{a})[x \mapsto \mathbf{0}] \Rightarrow \mathit{False}$ holds and condition 1(b)ii is met. ⊠

Following the same line of the previous two examples, we are able to show that the proposed format applies to all of the examples in Section 3.4.

Consider now Example 3.5.1. This example is interesting because, in order to meet condition 1a for the only axiom $\mathrm{RUN}_a \xrightarrow{a} \mathrm{RUN}_a$, we must choose J to be the set containing both of the rules (y) and $(\mathit{not}\text{-}y)$. Choosing J to be a singleton set

containing one of the rules is not enough, because neither

$$\models_G \mathsf{True} \Rightarrow (x \xrightarrow{a} \wedge\, y \xrightarrow{a})[x \mapsto \mathrm{RUN}_a]$$

nor

$$\models_G \mathsf{True} \Rightarrow (x \xrightarrow{a} \wedge\, y \xrightarrow{\!\!a\!\!\rightarrow})[x \mapsto \mathrm{RUN}_a]$$

hold. On the other hand, when $J = \{(y), (not - y)\}$, the entailment

$$\models_G \mathsf{True} \Rightarrow ((x \xrightarrow{a} \wedge\, y \xrightarrow{a}) \vee (x \xrightarrow{a} \wedge\, y \xrightarrow{\!\!a\!\!\rightarrow}))[x \mapsto \mathrm{RUN}_a]$$

holds and, moreover, $\mathrm{RUN}_a \cong_0 \mathrm{RUN}_a$, meeting condition 1(a)iiA. Therefore the proposed format can check Example 3.5.1, which cannot be dealt with by the format from Definition 3.3.5.

3.7 From zero to unit

In this section we reformulate the unit element format of [22] following the lines of Definition 3.6.4.

For the sake of clarity and completeness we repeat here the definition of unit-context equivalence from [22].

Definition 3.7.1 (Unit-context equivalence [22]) *Given sets $L, R \subseteq \Sigma \times \Sigma$ of pairs of binary function symbols and constants, $\overset{L,R}{\cong}$ is the smallest equivalence relation satisfying the following constraints, for each $s \in \mathbb{T}(\Sigma)$:*

1. $\forall (f, c) \in L.\ s \overset{L,R}{\cong} f(c, s)$, *and*
2. $\forall (g, c) \in R.\ s \overset{L,R}{\cong} g(s, c)$.

We say that two terms $s, t \in \mathbb{T}(\Sigma)$ are unit-context equivalent, *if $s \overset{L,R}{\cong} t$.*

Since the sets L and R are always clear from the context, we write $\cong$ in place of $\overset{L,R}{\cong}$.

Theorem 3.7.1 (Decidability of unit-context equivalence) *Let $L, R \subseteq \Sigma \times \Sigma$ be finite sets of pairs of binary function symbols and constants. Then, for all terms $t, u \in \mathbb{T}(\Sigma)$, it is decidable whether $t \overset{L,R}{\cong} u$ holds.*

Proof. Let L and R be given finite sets of pairs of binary operators and constants. Suppose that we are given two terms t and u and we want to check whether they

are unit-context equivalent. From t and u, construct the (undirected) graph $G(t, u)$ as follows.

The nodes in $G(t, u)$ are

- t, u and all their subterms,
- all constants mentioned in L or R, and
- all terms of the form $f(c, d)$ with $(f, c) \in L$ and $(f, d) \in R$.

The edges in $G(t, u)$ are given by items 1 and 2 in Definition 3.7.1. This graph is finite, since L and R are finite, and can be built effectively. Note that $G(u, t)$ and $G(t, u)$ are identical.

We claim that t is unit-context equivalent to u iff t can be reached from u in $G(t, u)$.

The proof of this claim is as follows. The right-to-left implication is immediate since each edge in $G(t, u)$ corresponds to an application of item 1 or item 2 in Definition 3.7.1. For the converse, we proceed by induction on the length of a shortest proof of $t \cong u$. If $t \cong u$ follows by reflexivity or by using item 1 or 2 in Definition 3.7.1 then t can be reached from u in $G(t, u)$ in zero steps or in one step, respectively. If $t \cong u$ follows by symmetry then the claim follows by the inductive hypothesis. Assume now that $t \cong u$ follows by transitivity. Then there is some term s such that $t \cong s$ (in one step) and $s \cong u$. By induction and the symmetry of reachability, s is reachable from t in $G(t, s)$ and s is reachable from u in $G(s, u)$. To see that u is reachable from t in $G(t, u)$, we now observe that s can be taken to be

- a subterm of t, if $t = f(c, s)$ for some $(f, c) \in L$ or $t = f(s, c)$ for some $(f, c) \in R$, or
- if t is a constant c, a term of one of the following forms for some constant d:
  - $f(c, d)$, where $(f, c) \in L$ and $(f, d) \in R$, or
  - $f(d, c)$, where $(f, c) \in R$ and $(f, d) \in L$.

In the former case, $G(t, s)$ and $G(s, u)$ are subgraphs of $G(t, u)$, and therefore t is reachable from u in $G(t, u)$ as claimed.

In the latter case, let, without loss of generality,

$$t = c \cong t_1 = f(d, c) \cong t_2 \cdots t_{n-1} \cong t_n = u \quad (n \geq 2)$$

be a shortest proof of $t \cong u$, where $(f, d) \in L$ and each of the intermediate equivalences is an instance of items 1 and 2 in Definition 21 or of their symmetric counterparts. Since the above is a shortest proof of $t \cong u$, we have that t_2 can be:

1. d, if $(f, c) \in R$,
2. $g(f(d, c), e)$, for some $(g, e) \in R$, or
3. $g(e, f(d, c))$, for some $(g, e) \in L$.

If $t_2 = d$ and $(f, c) \in R$, then $G(d, u)$ is a subgraph of $G(t, u)$ and d is reachable from $c = t$ in $G(t, u)$. In both the other cases, since the above is a shortest proof of $t \cong u$, we have that t_2 must be a subterm of u. Therefore, $G(t_2, u)$ is a subgraph of $G(t, u)$. Since $t_1 = f(d, c)$ and $c = t$ are also subterms of u, in all cases we have that t is reachable from u in $G(t, u)$.

It follows that both $G(t, s)$ and $G(s, u)$ are subgraphs of $G(t, u)$, and therefore t is reachable from u in $G(t, u)$, as claimed. ⊠

Definition 3.7.2 (Left- and right-aligned pairs for unit elements) *Given a GSOS language G, the sets L and R of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.*

1. *For each $(f, c) \in L$, the following conditions hold:*
   (a) *For each action $a \in \mathcal{L}$, there exists at least one deduction rule of the form*

$$\frac{\Phi \cup \{x_1 \xrightarrow{a} y_1\}}{f(x_0, x_1) \xrightarrow{a} t'},$$

   where
   i. $\models_G x_1 \xrightarrow{a} \Rightarrow \mathit{hyps}(\Phi)[x_0 \mapsto c]$, *and*
   ii. *one of the following cases holds:*
      A. *there are a premise $x_0 \xrightarrow{b} y \in \Phi$, for some $b \in \mathcal{L}$ and $y \in vars(t')$, and an axiom $c \xrightarrow{b} t$ such that $\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c, y to t and is the identity on all the other variables, or*
      B. *$\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c and is the identity on all the other variables.*

(b) *For each f-defining deduction rule*

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

one of the following cases holds:

i. $x_1 \xrightarrow{a} y_1 \in \Phi$ *for some variable* y_1 *and*

A. *either there is a premise* $x_0 \xrightarrow{b} y \in \Phi$, *for some* $b \in \mathcal{L}$ *and variable* $y \in vars(t')$, *such that* c *has a single axiom with label* b—*say,* $c \xrightarrow{b} t$—*and* $\sigma(t') \cong y_1$, *where* σ *is the substitution mapping* x_0 *to* c, y *to* t *and is the identity on all the other variables,*

B. *or* $\sigma(t') \cong y_1$, *where* σ *is the substitution mapping* x_0 *to* c *and is the identity on all the other variables.*

ii. $\models_G \mathsf{hyps}(\Phi)[x_0 \mapsto c] \Rightarrow \mathsf{False}$.

2. *The definition of right-aligned pairs of operators and constant symbols—that is, those such that* $(f, c) \in R$—*is symmetric and is not repeated here.*

For a function symbol f and a constant c, we call (f, c) left aligned *(respectively,* right aligned*) if* $(f, c) \in L$ *(respectively,* $(f, c) \in R$*).*

The following theorem states the correctness of the rule format defined above.

Theorem 3.7.2 *Let G be a GSOS language. Assume that L and R are the sets of left- and right-aligned function symbols according to Definition 3.7.2. For each $(f, c) \in L$, it holds that $f(c, x) \leftrightarrow x$. Symmetrically, for each $(f, c) \in R$, it holds that $f(x, c) \leftrightarrow x$.*

Section 3.11 contains the proof of Theorem 3.7.2.

Remark 3.7.3 *The constraint that $c \xrightarrow{b} t$ be the only c-defining axiom with label b in condition 1(b)iA of Definition 3.7.2 is necessary for the validity of Theorem 3.7.2. To see this, consider, for instance, the TSS over set of labels $\{a\}$ with constants $\mathbf{0}$, RUN_a (see Example 3.3.3) and c, and the binary operator $\|_{\mathcal{L}}$ defined in Example 3.4.1. The rules for the constant c are*

$$\frac{}{c \xrightarrow{a} c} \qquad \frac{}{c \xrightarrow{a} \mathbf{0}} \ .$$

Observe that the sets $L = \{(\|_{\mathcal{L}}, c)\}$ and $R = \emptyset$ would satisfy the conditions in Definition 3.7.2 if the uniqueness requirement were dropped from condition 1(b)iA. On the other hand,

$c \parallel_{\mathcal{L}} RUN_a$ is not bisimilar to RUN_a because

$$c \parallel_{\mathcal{L}} RUN_a \xrightarrow{a} \mathbf{0} \parallel_{\mathcal{L}} RUN_a \not\xrightarrow{a} \ ,$$

while RUN_a can only perform action a forever. Therefore c is not *a left unit element for $\parallel_{\mathcal{L}}$.* ⊠

The following result is a consequence of Theorems 3.6.1 and 3.7.1.

Theorem 3.7.3 *For GSOS languages, the sets L and R can be effectively constructed.*

The format for left and right unit elements proposed above is incomparable to the one offered in [22]. Indeed, the latter allows for complex terms as source of the conclusions and in the premises, which the GSOS format forbids. On the other hand, in condition 1a above, the set of premises Φ may contain several tests on the argument variable x_1, which is forbidden by the purely syntactic format in [22]. A concrete, albeit admittedly inexpressive, example of a TSS exploiting this feature is discussed below.

Example 3.7.4 *Consider a TSS, over the set of labels $\{a, b\}$, with constants RUN_a and RUN_b, and a binary function symbol f defined by the rules below.*

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

The constants RUN_a and RUN_b are both left unit elements for f. Indeed, every *closed term is a left unit element for f. This holds true because each closed term is bisimilar to one of the constants RUN_a and RUN_b. Therefore, every process is either able to perform initially an a-transition or is able to perform initially a b-transition, but never both.*

It is not hard to check that the sets $L = \{(f, RUN_a), (f, RUN_b)\}$ and $R = \emptyset$ satisfy the conditions in Definition 3.7.2. On the other hand, the format from [22] fails on this basic scenario since y is tested twice in the rules for f. ⊠

All the examples from the literature mentioned in [22] can be handled by the rule format presented in Definition 3.7.2. (Indeed, predicates can be dealt with within the proposed format along the lines discussed in Remark 3.6.6.) By way of illustration, we limit ourselves to discussing just a couple of examples addressed in [22].

Example 3.7.5 (Nondeterministic Choice) *Consider the classic nondeterministic choice operator from Milner's CCS [96] specified by the rules below, where $a \in \mathcal{L}$.*

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

The sets $R = L = \{(+, \mathbf{0})\}$ meet the constraints in Definition 3.7.2. Let us discuss the constraints relative to the set L. (The constraints for the set R can be checked using a similar reasoning.)

- *1a: Consider the second rule above. Here $\Phi = \emptyset$ and since $(\mathsf{hyps}(\Phi))[x \mapsto \mathbf{0}] = \mathsf{True}$, the entailment*

  $$\models_G y \xrightarrow{a} \Rightarrow (\mathsf{hyps}(\Phi))[x \mapsto \mathbf{0}]$$

  is trivially satisfied. Therefore condition 1(a)i is met. Note, moreover, that $y' \cong y'$. Therefore condition 1(a)ii is met too.

- *1b: Consider the first rule. Since $\mathbf{0} \not\xrightarrow{a}$, the entailment $\models_G (x \xrightarrow{a})[x \mapsto \mathbf{0}] \Rightarrow \mathsf{False}$ holds and condition 1(b)ii is met. Moreover, reasoning as above, we can easily check that the second rule above meets condition 1(b)i.* ⊠

Example 3.7.6 (Synchronous Parallel Composition) *Assume that a is the only action in $\mathcal{L}$. Consider the constant RUN_a and the synchronous parallel composition operator $\|_{\mathcal{L}}$ from Example 3.4.1. For ease of reference, we recall that $\|_{\mathcal{L}}$ is specified by the rule*

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_{\mathcal{L}} y \xrightarrow{a} x' \parallel_{\mathcal{L}} y'} \ .$$

Take $L = R = \{(\|_{\mathcal{L}}, RUN_a)\}$. These sets L and R meet the constraints in Definition 3.7.2. Let us discuss first the set L.

- *1a: Consider the rule above. Since $(x \xrightarrow{a})[x \mapsto RUN_a] = \mathsf{True}$, the entailment*

  $$\models_G y \xrightarrow{a} \Rightarrow (x \xrightarrow{a})[x \mapsto RUN_a]$$

  is trivially satisfied. Therefore condition 1(a)i is met. Note, moreover, that $x \xrightarrow{a} x'$ is a premise of the rule above. Since we can pick the axiom

  $$RUN_a \xrightarrow{a} RUN_a \ ,$$

the substitution σ that maps x and x' to RUN_a and that is the identity function on all the other variables is such that $\sigma(x' \parallel_{\mathcal{L}} y') \equiv RUN_a \parallel_{\mathcal{L}} y' \cong y'$. Therefore condition 1(a)iiA is met.

- *1b: Reasoning as above, we can easily check that rule above meets condition 1(b)iA in Definition 3.7.2.*

A similar reasoning shows that $(\parallel_{\mathcal{L}}, RUN_a)$ is also right aligned. ⊠

3.8 Conclusions

In this chapter we have provided two rule formats ensuring that certain constants in a language act as left or right zero elements for a set of binary operators. The format for left and right zero elements presented in Section 3.3 follows the techniques developed by some of the authors in [22], where a format for left and right unit elements was offered, but the actual details are rather different.

To overcome some drawbacks of that format, in Section 3.6.2 we reformulated it within the GSOS format, making use of the logic of initial transitions proposed in [7]. The new format is able to check relevant cases from the literature and some instances of zero elements left out by the format in Section 3.3.

Following the design of the revised format for zero elements, we also provided an alternative rule format for left and right unit elements. This format does not include advanced features such as complex terms in the source of the conclusions of rules, like the one in [22] instead does, but is still able to check relevant cases and basic unit elements not addressed by the format from [22].

We believe that the formats we propose in this chapter for GSOS languages are good candidates for mechanization in a tool-set for checking algebraic laws based on rule formats.

3.9 Proof of Theorem 3.3.2

The proof will rely on the following lemma, which can be shown by a straightforward induction on the definition of $\cong_0$.

Lemma 3.9.1 *For all $s, t \in \mathbb{T}(\Sigma)$, if $s \cong_0 t$ then $\sigma(s) \cong_0 \sigma(t)$, for each substitution σ.*

From Lemma 3.9.1, it trivially follows that, when t is a closed term, $s \cong_0 t$ implies $\sigma(s) \cong_0 t$ for each substitution σ. In the proof of Theorem 3.3.2 given below we make use of this observation.

Proof. (of Theorem 3.3.2)

We prove that $\cong_0$ is a bisimulation relation. The claim then follows since $f(c,p) \cong_0 p$ and $g(p,c') \cong_0 p$ for each closed term p, $(f,c) \in L$ and $(g,c') \in R$. In order to show that $\cong_0$ is a bisimulation it suffices to prove that whenever $p \cong_0 q$ then

- if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ for some q' such that $p' \cong_0 q'$,
- if $P\ p$ then $P\ q$, for each predicate P,
- if $q \xrightarrow{a} q'$ then $p \xrightarrow{a} p'$ for some p' such that $p' \cong_0 q'$, and
- if $P\ q$ then $P\ p$, for each predicate P.

We prove these statements by an induction on the definition of $\cong_0$. The cases that $p \cong_0 q$ is due to reflexivity, symmetry and transitivity of $\cong_0$ are trivial or follow easily using the inductive hypothesis. So, two relevant cases remain to be proved.

1. Suppose that $p \cong_0 q$ is due to $p \equiv c$ and $q \equiv f(c,p)$ for some $(f,c) \in L$.

   (a) Assume that $c \xrightarrow{a} p'$, for some $p' \in \mathbb{C}(\Sigma)$. This is because there exists an axiom $c \xrightarrow{a} p'$. We shall show that there exists a $p'' \in \mathbb{C}(\Sigma)$ such that $q \equiv f(c,p) \xrightarrow{a} p''$ and $p' \cong_0 p''$.

   By Definition 3.3.5, we have a deduction rule of the following form

$$\frac{\{x_0 \xrightarrow{a_i} t_i \mid i \in I\} \cup \{P_k\ x_0 \mid k \in K\} \cup \{x_0 \xrightarrow{a_j} \text{ or } \neg P_j\ x_0 \mid j \in J\}}{f(x_0,x_1) \xrightarrow{a} t'}$$

   where

   i. $x_1 \notin \{x_0\} \cup \bigcup_{i \in I} vars(t_i)$,

   ii. for each $j \in J$, there is no c-defining deduction rule with a_j or P_j as label (depending on whether the formula with index j is a transition or a predicate formula),

   iii. there exists a collection $\{P_k\ c \mid k \in K\}$ of c-defining axioms, and

   iv. there is a substitution σ such that $\sigma(x_0) = c$, $\{c \xrightarrow{a_i} \sigma(t_i) \mid i \in I\}$ is included in the collection of c-defining axioms, and $\sigma(t') \cong_0 p'$.

Since $x_1 \notin \{x_0\} \cup \bigcup_{i\in I} vars(t_i)$, we can extend σ to a closed substitution σ' mapping x_1 to p and all the variables not contained in $\{x_0, x_1\} \cup \bigcup_{i\in I} vars(t_i)$ to c. It is easy to see that the substitution σ' constructed in that fashion satisfies all the premises of the above rule. Thus, $f(c,p) \xrightarrow{a} \sigma'(t')$ is a provable transition. As $\sigma(t') \cong_0 p'$ by clause (iv) above and $\sigma'(\sigma(t')) \equiv \sigma'(t')$ by construction, Lemma 3.9.1 yields that $\sigma'(t') \cong_0 p'$, and we are done.

(b) Assume that $q \equiv f(c,p) \xrightarrow{a} q' \in C$, for some $q' \in \mathbb{C}(\Sigma)$. We shall show that there is some $p' \in \mathbb{C}(\Sigma)$ such that $c \xrightarrow{a} p'$ and $p' \cong_0 q'$.

By the proviso of the theorem, the transition $q \equiv f(c,p) \xrightarrow{a} q'$ must be proved using an f-defining rule. Therefore, it follows from constraint 1(b) in Definition 3.3.5 that the transition $q \equiv f(c,p) \xrightarrow{a} q'$ is due to a deduction rule of the following form

$$\frac{\Phi}{f(x_0,x_1) \xrightarrow{a} t'}$$

and a closed substitution σ such that $\sigma(x_0) \equiv c$, $\sigma(x_1) \equiv p$, $\sigma(t') \equiv q'$ and σ satisfies Φ.

Since σ satisfies Φ and $\sigma(x_0) \equiv c$, then the condition 1(b)ii does not apply and we fall in the case of constraint 1(b)i. Thus, by the proviso of the clause, we can identify an axiom $c \xrightarrow{a} p'$ for some p' such that $p' \cong_0 \sigma(t')$, and we are done.

(c) The two cases involving predicates, namely when Pc holds and $Pf(c,p)$ holds, follow the same lines.

2. Suppose that $p \cong_0 q$ is due to $p \equiv c$ and $q \equiv g(p,c)$ for some $(g,c) \in R$.

This case is similar to the previous one and we omit the details. ⊠

3.10 Proof of Theorem 3.6.2

We prove that the relation $\cong_0$ is a bisimulation. The claim then follows since $f(c,p) \cong_0 c$ and $g(p,c') \cong_0 c'$ for each closed term p, $(f,c) \in L$ and $(g,c') \in R$. To this end, we show that, when $p \cong_0 q$, the transfer conditions of Definition 3.2.6 are met by an induction on the definition of $\cong_0$. The cases that $p \cong_0 q$ is due to reflexivity,

symmetry and transitivity of $\cong_0$ are trivial or follow easily by induction. So, two relevant cases remain to be considered.

1. Suppose that $p \cong_0 q$ is due to $p \equiv c$ and $q \equiv f(c, q')$ for some $(f, c) \in L$ and closed term q'.

   (a) Assume that $c \xrightarrow{a} p' \in C$, for some $p' \in \mathbb{C}(\Sigma)$. This is because there exists an axiom $c \xrightarrow{a} p'$. We shall show that there exists a $p'' \in \mathbb{C}(\Sigma)$ such that $q \equiv f(c, q') \xrightarrow{a} p''$ and $p' \cong_0 p''$.

   From constraint 1(a)ii in Definition 3.6.4, we have a non-empty set J of deduction rules of the following form

   $$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

   such that

   i. $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(J)[x_0 \mapsto c]$, and

   ii. for each rule in J, one of the following cases holds:

      A. there is some variable $y \in \mathit{vars}(t')$ such that $x_0 \xrightarrow{a} y \in \Phi$ and $\sigma(t') \cong_0 p'$, where σ is the substitution mapping x_0 to c, y to p' and is the identity on all the other variables, or

      B. $\sigma(t') \cong_0 p'$, where σ is the substitution mapping x_0 to c and is the identity on all the other variables.

   Let σ' be an arbitrary closed substitution mapping x_0 to c and x_1 to q'. Since $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(J)[x_0 \mapsto c]$, we have that σ' satisfies the formula $\mathsf{hyps}(\Phi)[x_0 \mapsto c]$, where Φ is the set of premises of some rule r in the set J. If, for this rule r, we are in case 1(a)iiA above, let σ'' be the substitution that maps y to p' and acts like σ' on all the other variables. (In this case, the substitution σ'' satisfies the premise $x_0 \xrightarrow{b} y \in \Phi$.) Otherwise, let $\sigma'' = \sigma'$. By Lemma 3.6.1, we can construct a substitution σ''' that

   - 'extends' σ'' defined above,
   - maps x_1 to q' and
   - satisfies Φ.

Instantiating r with σ''' yields the transition $q \equiv f(c, q') \xrightarrow{a} \sigma'''(t')$. Since $\sigma(t') \cong_0 p'$, the term p' is a closed and σ''' 'extends' σ, Lemma 3.9.1 yields that $p' \cong_0 \sigma''(t')$, and we are done.

(b) Assume that $q \equiv f(c, q') \xrightarrow{a} p'' \in C$, for some $p'' \in \mathbb{C}(\Sigma)$. We shall show that $c \xrightarrow{a} p' \in C$, for some $p' \in \mathbb{C}(\Sigma)$ such that $p' \cong_0 p''$.

It follows from constraint 1b in Definition 3.6.4 that the transition $q \equiv f(c, q') \xrightarrow{a} p''$ is due to a deduction rule of the following form

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

and a closed substitution σ' such that $\sigma'(x_0) \equiv c$, $\sigma'(x_1) \equiv q'$, $\sigma'(t') \equiv p''$ and σ' satisfies Φ.

Since σ' satisfies Φ and $\sigma'(x_0) \equiv c$, condition 1(b)ii in Definition 3.6.4 cannot apply and we fall in the case of constraint 1(a)ii. Thus we can find an axiom $c \xrightarrow{a} p'$ and show that $\sigma'(t') \cong_0 p'$ reasoning as in the case above.

2. Suppose that $p \cong_0 q$ is due to $q \equiv g(p, c)$ for some $(g, c) \in R$.

This proof is similar to the one for the previous case and we omit the details. ⊠

3.11 Proof of Theorem 3.7.2

The proof relies on the following lemma proved in [22].

Lemma 3.11.1 *For all $s, t \in \mathbb{T}(\Sigma)$, if $s \cong t$ then $vars(s) = vars(t)$ and $\sigma(s) \cong \sigma(t)$, for each substitution σ.*

The proof of Theorem 3.7.2 is given below.

Proof. (of Theorem 3.7.2) We prove that $\cong$ is a bisimulation relation. The claim then follows since $f(c, p) \cong p$ and $g(p, c') \cong p$ for each closed term p, $(f, c) \in L$ and $(g, c') \in R$. We prove that whenever $p \cong q$ the transfer conditions of Definition 6 are met by an induction on the definition of $\cong$. The cases that $p \cong q$ is due to reflexivity, symmetry and transitivity of $\cong$ are trivial or follow easily by induction. So, two relevant cases remain to be proved.

1. Suppose that $p \cong q$ is due to $q \equiv f(c,p)$ for some $(f,c) \in L$.

   (a) Assume that $p \xrightarrow{a} p' \in C$, for some $p' \in \mathbb{C}(\Sigma)$. We shall show that there exists a $p'' \in \mathbb{C}(\Sigma)$ such that $q \equiv f(c,p) \xrightarrow{a} p''$ and $p' \cong p''$.

   From constraint 1a in Definition 3.7.2, we have that there exists a deduction rule of the following form

   $$\frac{\Phi \cup \{x_1 \xrightarrow{a} y_1\}}{f(x_0,x_1) \xrightarrow{a} t'},$$

   where

   i. $\models_G x_1 \xrightarrow{a} \Rightarrow \mathsf{hyps}(\Phi)[x_0 \mapsto c]$, and

   ii. one of the following cases holds:

      A. there are a premise $x_0 \xrightarrow{b} y \in \Phi$, for some $b \in \mathcal{L}$ and $y \in vars(t')$, and an axiom $c \xrightarrow{b} t$ such that $\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c, y to t and is the identity on all the other variables, or

      B. $\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c and is the identity on all the other variables.

   We now show that there exists a closed substitution σ' such that σ' satisfies Φ, $f(c,p) \xrightarrow{a} \sigma'(t')$ is a provable transition and $\sigma'(t') \cong p'$. Consider an arbitrary closed substitution σ'' mapping x_0 to c, x_1 to p and y_1 to p' and not precisely specified elsewhere at the moment. Such a substitution σ'' satisfies the premise $x_1 \xrightarrow{a} y$. If we are in case 1(a)iiA, let $\sigma''(y) \equiv t$, so that σ'' also satisfies the premise $x_0 \xrightarrow{b} y \in \Phi$.

   As σ'' satisfies the premise $x_1 \xrightarrow{a} y_1$, $\sigma''(x_0) \equiv c$ and $\models_G x_1 \xrightarrow{a} \Rightarrow \mathsf{hyps}(\Phi)[x_0 \mapsto c]$, we have that $\rightarrow_G, \sigma'' \models \mathsf{hyps}(\Phi)$. Therefore Lemma 3.6.1 yields a closed substitution σ' such that

   - $\sigma'(x_i) = \sigma''(x_i)$ for each $i \in \{0,1\}$,
   - $\sigma'(y_1) = \sigma''(y_1) = p'$,
   - $\sigma'(y) = \sigma''(y) = t$ if we are in case 1(a)iiA and
   - σ' satisfies Φ.

Instantiating the rule

$$\frac{\Phi \cup \{x_1 \xrightarrow{a} y_1\}}{f(x_0, x_1) \xrightarrow{a} t'},$$

with such a closed substitution σ' yields the transition

$$\sigma'(f(x_0, x_1)) \equiv f(c, p) \xrightarrow{a} \sigma'(t') \ .$$

Recall that $\sigma(t') \cong y_1$, where σ is either the substitution defined in case 1(a)iiA or 1(a)iiB. In both cases, by Lemma 3.11.1, we have that

$$\sigma'(t') = \sigma'(\sigma(t')) \cong \sigma'(y_1) = p'$$

and we are done.

(b) Assume that $q \equiv f(c, p) \xrightarrow{a} q' \in C$, for some $q' \in \mathbb{C}(\Sigma)$.

The transition $q \equiv f(c, p) \xrightarrow{a} q' \in C$ must be proved using an f-defining rule of the form

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

and a closed substitution σ' such that $\sigma'(x_0) \equiv c$, $\sigma'(x_1) \equiv p$, $\sigma'(t') \equiv q'$ and σ' satisfies Φ. Since σ' satisfies Φ and $\sigma'(x_0) \equiv c$, condition 1(b)ii in Definition 3.7.2 cannot apply and we fall in the case of constraint 1(b)i. Thus $x_1 \xrightarrow{a} y_1 \in \Phi$ for some variable y_1. As σ' satisfies Φ, it follows that $\sigma'(x_1) \equiv p \xrightarrow{a} \sigma'(y_1)$. We claim that $\sigma'(y_1) \cong q'$. To see that this claim does hold true, recall that, since constraint 1(b)i in Definition 3.7.2 is met,

i. either there is a premise $x_0 \xrightarrow{b} y \in \Phi$, for some $b \in \mathcal{L}$ and variable $y \in vars(t')$, such that c has a single axiom with label b—say, $c \xrightarrow{b} t$—and $\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c, y to t and is the identity on all the other variables,

ii. or $\sigma(t') \cong y_1$, where σ is the substitution mapping x_0 to c and is the identity on all the other variables.

In the former case, σ' satisfies the premise $x_0 \xrightarrow{b} y \in \Phi$. Therefore, $\sigma'(x_0) \equiv c \xrightarrow{b} t \equiv \sigma'(y)$, as $c \xrightarrow{b} t$ is the only c-defining axiom with label b. By Lemma 3.11.1, since $\sigma(t') \cong y_1$ holds, we have that

$$q' = \sigma'(t') = \sigma'(\sigma(t')) \cong \sigma'(y_1) = p'$$

and we are done.

The latter case is handled similarly.

2. Suppose that $p \cong q$ is due to $q \equiv g(p, c)$ for some $(g, c) \in R$.

   This case is similar to the previous case and we omit the details. ⊠

Chapter 4

Rule Formats for Distributivity

The white man knows how to make everything, but he does not know how to distribute it.
Sitting Bull.

4.1 Introduction

The syntax of a programming or specification language defines the collection of syntactically correct expressions, and its core is typically described formally using some variation on the notion of grammar. The semantics of a language associates a 'meaning' to each syntactically correct expression.

Over the last three decades, Structural Operational Semantics (SOS), see, e.g., [18, 109, 119, 65], has proven to be a powerful way to specify the semantics of programming and specification languages. In this approach to semantics, languages can be given a clear behaviour in terms of states and transitions, where the collection of transitions 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.

Designers of languages often have expected algebraic properties of language constructs in mind when defining a language. For example, one expects a sequential composition operator to be associative and, in the field of process algebra [24, 38, 77, 96], operators such as nondeterministic and parallel composition are often meant to be commutative and associative with respect to bisimilarity [115].

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 equivalence or preorder 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 semantic properties of the various operators in the language. An alternative approach is to ensure the validity of algebraic properties *a priori*, i.e., *by design*, using the so called *SOS rule formats* [21]. 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 by design. Not surprisingly, the definition of 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 as possible and, preferably, the syntactic constraints of the format should be algorithmically checkable.

The literature on SOS provides rule formats for basic algebraic properties of operators such as commutativity [110], associativity [49], idempotence [4] and the existence of unit and zero elements [11, 22]. The main advantage of this approach is that one is able to verify the desired property 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. These formats provide one with an insight as to the semantic nature of algebraic properties and its link to the syntax of SOS rules. Additionally, rule formats like those presented in the above-mentioned references may serve as a guideline for language designers who want to ensure, a priori, that the constructs under design enjoy certain basic algebraic properties.

In this work, we develop two rule formats guaranteeing that certain binary operators are left distributive with respect to others modulo bisimilarity. A binary operator $\circledast$ is *left distributive* with respect to a binary operator $\oplus$, modulo some notion of behavioural equivalence, whenever the following equation holds

$$(x \oplus y) \circledast z = (x \circledast z) \oplus (y \circledast z).$$

A classic example of left-distributivity law within the realm of process algebra is

$$(x + y) \mathbin{\underline{\|}} z = (x \mathbin{\underline{\|}} z) + (y \mathbin{\underline{\|}} z),$$

where '+' and '$\underline{\|}$' stand for nondeterministic choice and left merge, respectively, from [24, 38, 96]. (The reader may find many other examples in the remainder of this chapter.) Distributivity laws like the aforementioned one play a crucial role in (ground-)complete axiomatizations of behavioural equivalences over fragments of process algebras (see, e.g., the above-mentioned references and [5, 14, 15]), and their lack of validity with respect to choice-like operators is often the key to the nonexistence of finite (in)equational axiomatizations of behavioural semantics—see, for instance, [13, 16, 100, 101].

The first rule format we present is the simplest of the two, but suffices to handle many examples from the literature. The second rule format has more complex syntactic conditions and can handle left-distributivity laws that are outside the scope of the former format. In both rule formats, for the sake of simplicity, the $\oplus$ operator 'behaves like' some form of nondeterministic choice operator. Both rule formats are based on syntactic conditions that are decidable over finite language specifications. Interestingly, the syntactic conditions of the second rule format are based on a notion of distributivity compliance, which is itself built on rule formats for other algebraic properties such as idempotence.

We provide a wealth of examples showing that the validity of several left-distributivity laws from the literature on process algebras can be proved using the two rule formats. Moreover, in Section 4.6 we argue that the two rule formats can be applied just as well to show distributivity laws of the form $f(x \oplus y) = f(x) \oplus f(y)$ involving a *unary* operator f.

We also offer some impossibility results concerning the validity of the left-distributivity law. Unlike previous results about rule formats for algebraic properties, these theorems allow one to recognize when the left-distributivity law is guaranteed *not* to hold. When designing operational specifications for operators that are intended to satisfy a left-distributivity law, a language designer might also benefit from considering these kinds of negative results. To our knowledge this type of result does not have any precursor in the field of rule formats. Hitherto, all rule formats aimed at providing sufficient conditions for establishing semantic properties, whereas the above-mentioned results are the first ones that offer *necessary syntactic conditions* for some semantic property to hold.

Roadmap of the chapter The chapter is organized as follows. Section 4.2 reviews some standard definitions from the theory of SOS that will be used in the remainder of this study. Section 4.3 presents our two rule formats guaranteeing that a binary operator $\circledast$ is left distributive with respect to a binary operator $\oplus$ modulo bisimilarity. The first rule format and some examples of its application are presented in Section 4.3.2. In Section 4.3.3, we introduce the second rule format, which extends the first rule format and can treat more examples. In order to ease its application, we simplify the checks in the second rule format in Section 4.4 and summarize the simplifications in a tabular form. Examples that can be handled using the second rule format (even by using the simplified checks in Section 4.4) are offered in Section 4.5. We apply the two rule formats to show left-distributivity laws involving unary operators in Section 4.6. Some impossibility results concerning the validity of the left-distributivity law are offered in Section 4.7. We conclude the chapter with a discussion of its contributions and of lines for future research in Section 4.8 . In order to increase the readability of the main body of the chapter, the proofs of the main technical results have been collected in sections that follow Section 4.8.

4.2 Preliminaries

In this section we recall some standard definitions from the theory of SOS. We refer the readers to, e.g., [18] and [109] for more information.

4.2.1 Transition system specifications and bisimilarity

Definition 4.2.1 (Signatures, terms and substitutions) *We let V denote an infinite set of variables and use $x, x', x_i, y, y', y_i, \ldots$ to range over elements of V. 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 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, u, possibly subscripted and/or superscripted, 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., terms*

t such that $vars(t) = \emptyset$. We use $p, q, p', p_i, \ldots$ 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 and write $\sigma(t)$ for the result of applying the substitution σ to the term t. If the range of a substitution is included in $\mathbb{C}(\Sigma)$, we say that it is a* closed substitution. *For a substitution σ, a sequence $x_1, \ldots, x_n$ of distinct variables and a sequence $t_1, \ldots, t_n$ of terms, we write*

$$\sigma[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$$

for the substitution that maps x_i to t_i, for each $1 \leq i \leq n$, and each variable $x \notin \{x_1, \ldots, x_n\}$ to $\sigma(x)$. Similarly, we write $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ for a substitution that maps x_i to t_i, for each $1 \leq i \leq n$, and acts like the identity function on all the other variables.

Definition 4.2.2 (Transition system specification) *A* transition system specification *(TSS) $\mathcal{T}$ is a triple $(\Sigma, \mathcal{L}, D)$ where*

- *Σ is a signature.*
- *$\mathcal{L}$ is a set of* labels *(or* actions*) ranged over by a, b, l. If $l \in \mathcal{L}$ and $t, t' \in \mathbb{T}(\Sigma)$, we say that $t \xrightarrow{l} t'$ is a* positive transition formula *and $t \overset{l}{\nrightarrow}$ is a* negative transition formula*. Such formulae are called t-*testing*. A transition formula (or just formula), typically denoted by ϕ or ψ, is either a negative transition formula or a positive one.*
- *D is a set of* deduction rules*, i.e., tuples 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 write $vars(\Phi)$ to denote the set of variables appearing in a set of formulae Φ, and $vars(r)$ to denote the set of variables appearing in a deduction rule r. We say that a formula or a deduction rule is closed *if all of its terms are closed. A deduction rule is t-*testing*, or* tests *t, if one of its premises is t-testing. Substitutions are also extended to formulae and sets of formulae in the natural way. For a rule r and a substitution σ, the rule $\sigma(r)$ is called a substitution instance of r. A set of positive closed formulae is called a* transition relation*.*

We often refer to a positive transition formula $t \xrightarrow{l} t'$ as a *transition* with t being its *source*, l its *label*, and t' its *target*. A deduction rule (Φ, ϕ) is typically written as $\frac{\Phi}{\phi}$. For the sake of consistency with SOS specifications of specific operators in the literature, in examples we use $\frac{\phi_1 \ldots \phi_n}{\phi}$ in lieu of $\frac{\{\phi_1, \ldots, \phi_n\}}{\phi}$.

An *axiom* is a deduction rule with an empty set of premises. We write $\frac{}{\phi}$ for an axiom with ϕ as its conclusion, and often abbreviate this notation to ϕ when this causes no confusion.

Definition 4.2.3 *Given a rule d of the form*

$$\frac{\Phi}{f(t_1, \ldots, t_n) \xrightarrow{a} t},$$

we say that

- *d is f-*defining*, and write $op(d) = f$,*
- *d is a-*emitting*,*
- *$toc(d) = t$, the target of the conclusion of d, and*
- *$hyps(d) = \Phi$, the set of premises of d.*

We also denote by $D(f, a)$ the set of a-emitting and f-defining rules in a set of deduction rules D.

Example 4.2.4 (Choice operators) *The choice operator from [96] is defined by the following rules, where a ranges over the set of actions:*

$$(chl_a)\ \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad (chr_a)\ \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}.$$

For each action a, the rules (chl_a) and (chr_a) are a-emitting and $+$-defining. For rule (chl_a), we have that $toc(chl_a) = x'$ and $hyps(chl_a) = \{x \xrightarrow{a} x'\}$.

For illustrative purposes in the remainder of the chapter the following 'choice' operators are introduced. The left choice operator $+_l$ is defined by the rules chl_a (there is one such rule for each action a). Symmetrically, the right choice operator $+_r$ is defined by the rules chr_a. (Again, there is one such rule for each action a.)

$$(chl_a)\ \frac{x \xrightarrow{a} x'}{x +_l y \xrightarrow{a} x'} \qquad (chr_a)\ \frac{y \xrightarrow{a} y'}{x +_r y \xrightarrow{a} y'}$$

The meaning of a TSS is defined by the following notion of least three-valued stable model. To define this notion, we need two auxiliary definitions, namely provable transition rules and consistency, which are given below.

Definition 4.2.5 (Provable transition rules) *A closed deduction rule is called a* transition rule *when it is of the form* $\frac{N}{\phi}$*, where* N *is a set of* negative formulae*. A TSS* $\mathcal{T}$ proves $\frac{N}{\phi}$*, denoted by* $\mathcal{T} \vdash \frac{N}{\phi}$*, when there is a well-founded upwardly branching tree with closed formulae as nodes and of which*

- *the root is labelled by* ϕ*;*
- *if a node is labelled by* ψ *and the labels of the nodes directly above it form the set* K *then:*
  - ψ *is a negative formula and* $\psi \in N$*, or*
  - ψ *is a positive formula and* $\frac{K}{\psi}$ *is a substitution instance of a deduction rule in* $\mathcal{T}$*.*

We often write $\mathcal{T} \vdash \phi$ *in lieu of* $\mathcal{T} \vdash \frac{\emptyset}{\phi}$*.*

Definition 4.2.6 (Contradiction and consistency) *The formula* $t \xrightarrow{l} t'$ *is said to* contradict $t \xnrightarrow{l}$*, and vice versa. For two sets* Φ *and* Ψ *of formulae,* Φ contradicts Ψ *when there is a* $\phi \in \Phi$ *that contradicts a* $\psi \in \Psi$*. We write* $\Phi \nvDash \Psi$*, read '*Φ *is consistent with* Ψ*', when* Φ *does not contradict* Ψ*.*

It immediately follows from the above definition that contradiction and consistency are symmetric relations on (sets of) formulae. We now have all the necessary ingredients to define the semantics of TSSs in terms of three-valued stable models [123].

Definition 4.2.7 (Three-valued stable model) *A pair* (C, U) *of disjoint sets of positive closed transition formulae is called a* three-valued stable model *for a TSS* $\mathcal{T}$ *when the following conditions hold:*

- *for each* $\phi \in C$*, there is a set* N *of negative formulae such that* $\mathcal{T} \vdash \frac{N}{\phi}$ *and* $C \cup U \nvDash N$*, and*
- *for each* $\phi \in C \cup U$*, there is a set* N *of negative formulae such that* $\mathcal{T} \vdash \frac{N}{\phi}$ *and* $C \nvDash N$*.*

C stands for Certainly *and U for* Unknown*; the third value is determined by the formulae not in* $C \cup U$*. The* least *three-valued stable model is a three-valued stable model that is the least one with respect to the (information-theoretic) ordering on pairs of sets of formulae defined as* $(C, U) \leq (C', U')$ *iff* $C \subseteq C'$ *and* $U' \subseteq U$*. We say that* $\mathcal{T}$ *is* complete *when for its least three-valued stable model it holds that* $U = \emptyset$*. In a complete TSS, we say that a closed substitution* σ *satisfies a set of formulae* Φ *if* $\sigma(\phi) \in C$*, for each positive formula*

$\phi \in \Phi$, and $C \vDash \{\sigma(\phi)\}$, for each negative formula $\phi \in \Phi$. If a TSS is complete, we often also write $p \xrightarrow{l} p'$ in lieu of $(p \xrightarrow{l} p') \in C$, and $p \not\xrightarrow{l}$ when there is no p' such that $p \xrightarrow{l} p'$.

In what follows, we shall tacitly restrict ourselves to considering only complete TSSs.

Definition 4.2.8 (Bisimulation and bisimilarity [96, 115]) *Let $\mathcal{T}$ be a transition system specification with signature Σ and label set $\mathcal{L}$. A relation $\mathcal{R} \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* bisimulation relation *if and only if $\mathcal{R}$ is symmetric and, for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ and $l \in \mathcal{L}$,*

$$(p_0 \,\mathcal{R}\, p_1 \wedge \mathcal{T} \vdash p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (\mathcal{T} \vdash p_1 \xrightarrow{l} p_1' \wedge p_0' \,\mathcal{R}\, p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called bisimilar*, denoted by $p_0 \leftrightarroweq p_1$, when there exists a bisimulation relation $\mathcal{R}$ such that $p_0 \,\mathcal{R}\, p_1$.*

Bisimilarity is extended to open terms by requiring that $s, t \in \mathbb{T}(\Sigma)$ are bisimilar when $\sigma(s) \leftrightarroweq \sigma(t)$ for each closed substitution $\sigma : V \to \mathbb{C}(\Sigma)$.

4.3 The left-distributivity rule formats

In this section, we present two rule formats guaranteeing that a binary operator $\circledast$ is left distributive with respect to a binary operator $\oplus$ modulo bisimilarity. The first rule format is the simplest of the two, but nevertheless suffices to handle many examples from the literature. The second rule format has more complex conditions and can handle left-distributivity laws that are outside the scope of the former format.

Definition 4.3.1 (Left-distributivity law) *We say that a binary operator $\circledast$ is* left distributive *with respect to a binary operator $\oplus$ (modulo bisimilarity) if the following equality holds:*

$$(x \oplus y) \circledast z \leftrightarroweq (x \circledast z) \oplus (y \circledast z). \tag{4.1}$$

For all closed terms p, q, r, proving the algebraic law (4.1) involves two proof obligations:

- **Firability:** ensuring that $(p \oplus q) \circledast r \xrightarrow{a}$ if, and only if, $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a}$, for each action a;

- **Matching conclusions:** ensuring that, for each closed term p_1, if $(p \oplus q) \circledast r \xrightarrow{a} p_1$, then there exists some closed term p_2 such that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} p_2$ and $p_1 \leftrightarrows p_2$, and vice versa.

Logically, the 'firability condition' is implied by the 'matching-conclusion condition'. However, since the two rule formats we shall present in what follows use the same idea to guarantee the former condition, and differ in how they guarantee the existence of matching conclusions up to bisimilarity, we prefer to consider the two conditions separately. To our mind, this also leads to a clearer presentation of the ideas underlying the rule formats. In what follows, we first explain how we achieve the 'firability condition', and then we discuss how the two different rule formats guarantee the 'matching-conclusion condition'.

4.3.1 The firability condition

We begin by introducing the conditions on sets of rules for two binary operators $\circledast$ and $\oplus$ that we shall use to guarantee the firability condition for them. First of all, we present syntactic constraints on the rules for those operators that we shall use throughout the remainder of the chapter.

Definition 4.3.2 *We say that a deduction rule is of* the form (R1) *when it has the structure*

$$\frac{\Phi_y}{x \circledast y \xrightarrow{a} t} \quad \textit{or} \quad \frac{\{x \xrightarrow{a} x'\} \cup \Phi_y}{x \circledast y \xrightarrow{a} t}.$$

where

- *the variables x, x', y are pairwise distinct, and*
- *Φ_y is a (possibly empty) set of (positive or negative) y-testing formulae such that $x, x' \notin vars(\Phi_y)$.*

A deduction rule is of the form (R2) *when it has the structure*

$$\frac{\{x \xrightarrow{a} x'\}}{x \oplus y \xrightarrow{a} t} \quad \textit{or} \quad \frac{\{y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} t} \quad \textit{or} \quad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} t}.$$

where the variables x, x', y, y' are pairwise distinct

A rule of the form (*R*1) *or* (*R*2) *is* non-left-inheriting *if* $x \notin vars(t)$*, that is, if* x *does not appear in the target of the conclusion of the rule. An operation* f *specified by rules of the form* (*R*1) *or* (*R*2) *is* non-left-inheriting *if so are all of the* f*-defining rules.*

Definition 4.3.3 (Firability constraint) *Given a TSS* T*, let* $\circledast$ *and* $\oplus$ *be binary operators in the signature of* T*. For each action* a*, we write* $Fire(\circledast, \oplus, a)$ *whenever the following conditions are met:*

- *if* $D(\circledast, a) \neq \emptyset$ *then* $D(\oplus, a) \neq \emptyset$*,*
- *each* $d \in D(\circledast, a)$ *is of the form* (*R*1)*, and*
- *each* $d \in D(\oplus, a)$ *is of the form* (*R*2)*.*

Remark 4.3.4 *Note that the first constraint in the definition of* $Fire(\circledast, \oplus, a)$ *is asymmetric, as it only requires that if there is a* $\circledast$*-defining* a*-emitting rule, then there should also be some* $\oplus$*-defining* a*-emitting rule. As will become clear from Examples 4.6.1–4.6.3, amongst others, this leads to a widely applicable rule format for left distributivity.*

Example 4.3.5 *Recall the choice operators* $+$*,* $+_l$ *and* $+_r$ *presented in Example 4.2.4. As our readers can easily check,* $Fire(f, g, a)$ *holds for each action* a *and for all* $f, g \in \{+, +_l, +_r\}$*.*

The firability constraint in Definition 4.3.3 is sufficient to guarantee the aforementioned firability condition.

Theorem 4.3.6 (Firability Theorem) *Given a TSS* T*, let* $\circledast$ *and* $\oplus$ *be binary operators from the signature of* T*. Suppose that* $Fire(\circledast, \oplus, a)$ *holds for some action* a*. Then,*

$$(p \oplus q) \circledast r \xrightarrow{a} \text{ if, and only if, } (p \circledast r) \oplus (q \circledast r) \xrightarrow{a},$$

for all closed terms p, q, r*.*

Proof. See Section 4.9.

The import of Theorem 4.3.6 is that, when proving the validity of (4.1), we can guarantee the firability condition for action a just by showing that $Fire(\circledast, \oplus, a)$ holds. Theorem 4.3.6 underlies the soundness of both the rule formats we present in what follows.

The reader will have already noticed that the rule form (*R*1) does not place any restriction on tests for the variable y. This is possible because the second argument of the terms $(p \oplus q) \circledast r$, $p \circledast r$ and $q \circledast r$ is always the same, i.e., the term r. This means that, for each $\circledast$-defining rule, the same tests performed on the second argument on one side of (4.1) are performed on the other. Roughly speaking, one

side of (4.1) may fire as much as the other does, insofar the second argument is concerned.

4.3.2 The matching-conclusion condition

Theorem 4.3.6 tells us that any rule format, whose constraints imply condition Fire($\circledast, \oplus, a$) for each action a, guarantees the validity of (4.1) provided that the matching-conclusion condition is met. Intuitively, in order to guarantee syntactically that the matching-conclusion condition is satisfied, the targets of the conclusions of $\circledast$-defining and $\oplus$-defining rules should 'match' when those operators are used in the specific contexts of the left- and the right-hand sides of (4.1). In what follows, we shall examine two different ways of ensuring the above-mentioned 'match' of the targets of the conclusions of $\circledast$-defining and $\oplus$-defining rules. The first relies on assuming that the targets of the conclusions of $\oplus$-defining rules are target variables of premises of rules of the form ($R2$). The resulting rule format, which we present in this section, is based on easily checkable syntactic constraints and covers a large number of left-distributivity laws from the literature. However, there are some examples of left-distributivity axioms that cannot be shown valid using that format. In order to be able to deal with more cases, including thise that might be presented in the literature in the future, in Section 4.3.3 we propose a more complex rule format in which the 'match' of the targets of the conclusions of $\circledast$-defining and $\oplus$-defining rules is performed by means of a powerful 'compliance relation'.

The first rule format

The first rule format that we present deals with examples of left distributivity with respect to operators whose semantics is given by rules of the form ($R2$) that, like those for the choice operators we mentioned in Example 4.2.4, have target variables of premises as targets of their conclusions. The following definition presents the syntactic constraints of the rule format.

Definition 4.3.7 (First rule format) *Let $\mathcal{T}$ be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of $\mathcal{T}$. We say that the rules for $\circledast$ and $\oplus$ are in the* first rule format for left distributivity *if the following conditions are met:*

1. *Fire($\circledast, \oplus, a$) holds for each action a,*

2. $\circledast$ *is non-left-inheriting,*
3. *each* $\oplus$*-defining rule has a target variable of one of its premises as target of its conclusion and*
4. *for each action* a*, either there is no* a*-emitting and* $\oplus$*-defining rule that tests both* x *and* y*, or if some* a*-emitting and* $\circledast$*-defining rule tests its left argument* x *then so do all* a*-emitting and* $\circledast$*-defining rules.*

Theorem 4.3.8 (Left distributivity over choice-like operators) *Let* $\mathcal{T}$ *be a TSS, and let* $\circledast$ *and* $\oplus$ *be binary operators in the signature of* $\mathcal{T}$*. Assume that the rules for* $\circledast$ *and* $\oplus$ *are in the* first rule format for left distributivity*. Then*

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z).$$

Proof. We show the following two claims, where p, q, r, s are arbitrary closed terms and a is any action:

1. If $(p \oplus q) \circledast r \xrightarrow{a} s$ then $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$.
2. If $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$ then $(p \oplus q) \circledast r \xrightarrow{a} s$.

In the proof of the former claim, we use the first condition in Definition 4.3.3. This condition is not used in the proof of the latter claim. On the other hand, the proof of the latter statement uses condition 4 in Definition 4.3.7, which is not used in the proof of the former claim. The full proof may be found in Section 4.10.

Remark 4.3.9 *Condition 4 in Definition 4.3.7 cannot be dropped without jeopardizing the soundness of the rule format for left distributivity proved in the above theorem. To see this, consider the operations* $\oplus$ *and* $\circledast$ *with rules*

$$\frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} x'} \qquad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \circledast y \xrightarrow{a} x' \circledast y} \qquad \frac{\{y \xrightarrow{a} y'\}}{x \circledast y \xrightarrow{a} y'}.$$

The above rules satisfy all the conditions in Definition 4.3.7 apart from condition 4. Now, let a *be a constant with rule* $a \xrightarrow{a} \mathbf{0}$*, where* $\mathbf{0}$ *is a constant with no rules. As our readers can easily check,*

$$(a \circledast a) \oplus (\mathbf{0} \circledast a) \underline{\leftrightarrows} (a \oplus \mathbf{0}) \circledast a.$$

Indeed, the term $(a \circledast a) \oplus (\mathbf{0} \circledast a)$ *can perform a sequence of two* a*-labelled transitions, whereas* $(a \oplus \mathbf{0}) \circledast a$ *cannot because* $a \oplus \mathbf{0}$ *affords no transitions.*

Examples of application of the first rule format

Theorem 4.3.8 provides us with a simple, yet rather powerful, syntactic condition in order to infer left-distributivity laws for operators like + and $+_l$. Many of the common left-distributivity laws are automatically derived from Theorem 4.3.8, as witnessed by the examples we now proceed to discuss.

Example 4.3.10 (Left merge and interleaving parallel composition) *The operational semantics of the classic left-merge and interleaving parallel composition operators [24, 37, 38, 96] is given by the rules below:*

$$\frac{x \xrightarrow{a} x'}{x \mathbin{\underline{\|}} y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}.$$

Note that the rules for the left-merge operator $\mathbin{\underline{\|}}$ and those for any of $+$, $+_l$ and $+_r$ satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following laws.

$$\begin{aligned}(x + y) \mathbin{\underline{\|}} z &\leftrightarrows (x \mathbin{\underline{\|}} z) + (y \mathbin{\underline{\|}} z)\\ (x +_l y) \mathbin{\underline{\|}} z &\leftrightarrows (x \mathbin{\underline{\|}} z) +_l (y \mathbin{\underline{\|}} z)\\ (x +_r y) \mathbin{\underline{\|}} z &\leftrightarrows (x \mathbin{\underline{\|}} z) +_r (y \mathbin{\underline{\|}} z)\end{aligned}$$

Observe that the equalities

$$\begin{aligned}(x +_l y) \parallel z &\leftrightarrows (x \parallel z) +_l (y \parallel z) \text{ and}\\ (x +_r y) \parallel z &\leftrightarrows (x \parallel z) +_r (y \parallel z)\end{aligned}$$

are sound. However, their soundness cannot *be shown using Theorem 4.3.8, since the parallel composition operator $\parallel$ does not satisfy condition 2 in Definition 4.3.7. Indeed, x occurs in the target of the conclusion of the second rule for $\parallel$.*

Example 4.3.11 (Synchronous parallel composition) *Consider the synchronous parallel composition from CSP [77, 76][1] specified by the rules below, where a ranges over the set of actions:*

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_{\mathcal{L}} y \xrightarrow{a} x' \parallel_{\mathcal{L}} y'}.$$

[1] In [77], Hoare uses the symbol $\parallel$ to denote the synchronous parallel composition operator. Here we use that symbol for interleaving parallel composition.

Note that the rules for the synchronous parallel composition operator and those for any of $+$, $+_l$ and $+_r$ satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following laws.

$$\begin{aligned}(x + y) \parallel_{\mathcal{L}} z &\leftrightarrows (x \parallel_{\mathcal{L}} z) + (y \parallel_{\mathcal{L}} z)\\ (x +_l y) \parallel_{\mathcal{L}} z &\leftrightarrows (x \parallel_{\mathcal{L}} z) +_l (y \parallel_{\mathcal{L}} z)\\ (x +_r y) \parallel_{\mathcal{L}} z &\leftrightarrows (x \parallel_{\mathcal{L}} z) +_r (y \parallel_{\mathcal{L}} z)\end{aligned}$$

Example 4.3.12 (Join and '/' operators) *Consider the join operator $\bowtie$ from [28] and the 'hourglass' operator $/$ from [5] specified by the rules below, where a, b range over the set of actions:*

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \bowtie y \xrightarrow{a} x' \mp y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x / y \xrightarrow{a} x' / y'},$$

where $\mp$ denotes the delayed choice operator from [28]. (The operational specification of the delayed choice operator is immaterial for the analysis of this example.) The above rules and those for any of $+$, $+_l$ and $+_r$ satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following laws, where $\circledast \in \{\bowtie, /\}$.

$$\begin{aligned}(x + y) \circledast z &\leftrightarrows (x \circledast z) + (y \circledast z)\\ (x +_l y) \circledast z &\leftrightarrows (x \circledast z) +_l (y \circledast z)\\ (x +_r y) \circledast z &\leftrightarrows (x \circledast z) +_r (y \circledast z)\end{aligned}$$

Example 4.3.13 (Disrupt) *Consider the following disrupt operator $\blacktriangleright$ [26, 46] with rules*

$$\frac{x \xrightarrow{a} x'}{x \blacktriangleright y \xrightarrow{a} x' \blacktriangleright y} \qquad \frac{y \xrightarrow{a} y'}{x \blacktriangleright y \xrightarrow{a} y'}.$$

The above rules and those for any of $+$, $+_l$ and $+_r$ satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following laws.

$$\begin{aligned}(x + y) \blacktriangleright z &\leftrightarrows (x \blacktriangleright z) + (y \blacktriangleright z)\\ (x +_l y) \blacktriangleright z &\leftrightarrows (x \blacktriangleright z) +_l (y \blacktriangleright z)\\ (x +_r y) \blacktriangleright z &\leftrightarrows (x \blacktriangleright z) +_r (y \blacktriangleright z)\end{aligned}$$

Example 4.3.14 (Unless operator) *The unless operator $\triangleleft$ from [27] and the operator Δ from [5, page 23] are specified by the rules*

$$\frac{x \xrightarrow{a} x' \quad y \overset{b}{\nrightarrow} \text{for } a < b}{x \triangleleft y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} x' \quad y \overset{b}{\nrightarrow} \text{for } a < b}{x \,\Delta\, y \xrightarrow{a} \theta(x')},$$

where $<$ is an irreflexive partial order over the set of actions and θ denotes the priority operator from [27]. (The operational specification of the priority operator is immaterial for the analysis of this example.) The above rules and those for any of $+$, $+_l$ and $+_r$ satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following laws, where $\circledast \in \{\triangleleft, \Delta\}$.

$$\begin{aligned} (x + y) \circledast z &\leftrightarrows (x \circledast z) + (y \circledast z) \\ (x +_l y) \circledast z &\leftrightarrows (x \circledast z) +_l (y \circledast z) \\ (x +_r y) \circledast z &\leftrightarrows (x \circledast z) +_r (y \circledast z) \end{aligned}$$

Example 4.3.15 (Interplay between the choice operators) *Consider the choice operators $+$, $+_l$ and $+_r$ from Example 4.2.4. The rules for any of the nine combinations of those operators satisfy the constraints of the first rule format for left distributivity. Therefore, Theorem 4.3.8 yields the validity of the following law, where $\oplus, \circledast \in \{+, +_l, +_r\}$.*

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z)$$

For example, as an instance of that family of equalities, we obtain the following 'self left-distributivity law' for any $\oplus \in \{+, +_l, +_r\}$:

$$(x \oplus y) \oplus z \leftrightarrows (x \oplus z) \oplus (y \oplus z).$$

As we show in Section 4.6, our first rule format for left distributivity can also be used to derive distributivity laws involving unary $\circledast$ operators.

4.3.3 The second left-distributivity format

As witnessed by the above-mentioned examples, the rule format introduced in Definition 4.3.7 can handle many of the common left-distributivity laws from the literature. However, as we mentioned in Example 4.3.10, that rule format is *not*

general enough to prove the validity of, e.g., the left-distributivity law

$$(x +_l y) \parallel z \leftrightarrow (x \parallel z) +_l (y \parallel z).$$

It is instructive to see why the equality

$$(p +_l q) \parallel r \leftrightarrow (p \parallel r) +_l (q \parallel r)$$

holds for all p, q, r. The terms that can be reached from $(p +_l q) \parallel r$ via an a-labelled transition have one of the two following forms:

- $p' \parallel r$, for some p' such that $p \xrightarrow{a} p'$ or
- $(p +_l q) \parallel r'$, for some r' such that $r \xrightarrow{a} r'$.

On the other hand, the terms that can be reached from $(p \parallel r) +_l (q \parallel r)$ via an a-labelled transition are of the form

- $p' \parallel r$, for some p' such that $p \xrightarrow{a} p'$ or
- $p \parallel r'$, for some r' such that $r \xrightarrow{a} r'$.

The first of those possible forms is identical to the first form of a possible derivative of $(p +_l q) \parallel r$. However, the second form—viz. $p \parallel r'$, for some r' such that $r \xrightarrow{a} r'$—matches $(p +_l q) \parallel r'$ only up to one application of the equation

$$x +_l y = x,$$

which is sound modulo bisimilarity, from left to right. This rewriting can be performed in the context of $\parallel$ since the rules for the interleaving parallel composition operator given in Example 4.3.10 are in de Simone format [52], which is one of the congruence formats for bisimilarity—see, for instance, the survey articles [18, 109].

The above discussion motivates the development of a generalization of the rule format we presented in Definition 4.3.7. The main idea behind this more powerful rule format is to weaken the constraints for ensuring the 'matching-conclusion condition', so that terms that are targets of transitions from $(p \oplus q) \circledast r$ and $(p \circledast r) \oplus (q \circledast r)$ need only be equal up to the application of some equation, whose validity modulo bisimilarity can be justified 'syntactically', in a context consisting of operations that preserve bisimilarity. Of course, the resulting definition of the rule format depends on the set of equations that one is allowed to use. Indeed, one can obtain more powerful rule formats by simply extending the collection of

allowed equations. Therefore, what we now present can be seen as a template for rule formats guaranteeing the validity of left-distributivity equations of the form (4.1). Our definition of the second rule format is based on a rewriting relation over terms that is sufficient to handle the examples from the literature we have met so far. The rewriting relation we present below can, however, be easily strengthened by adding more rewriting rules, provided their soundness with respect to bisimilarity can be 'justified syntactically'. (See the paragraphs after Definition 4.3.16 and Remark 4.3.23 for a brief discussion of extensions of the proposed rule format.)

Definition 4.3.16 (The rewriting relation $\rightsquigarrow$) *Let $\mathcal{T} = (\Sigma, \mathcal{L}, D)$ be a TSS.*

1. *The relation $\rightsquigarrow$ is the least binary relation over $\mathbb{T}(\Sigma)$ that satisfies the following clauses, where we use $t \leftrightsquigarrow t'$ as a short-hand for $t \rightsquigarrow t'$ and $t' \rightsquigarrow t$:*
   - $t \rightsquigarrow t$,
   - $f(t,t) \leftrightsquigarrow t$, *if $\mathcal{T}$ is in idempotence format with respect to f from [4],*
   - $C[t] \rightsquigarrow C[t']$, *if $t \rightsquigarrow t'$ and $\mathcal{T}$ is in a congruence format for $\leftrightarrows$,*
   - $t_1 +_l t_2 \rightsquigarrow t_1$, *if $+_l \in \Sigma$, and*
   - $t_1 +_r t_2 \rightsquigarrow t_2$, *if $+_r \in \Sigma$.*
2. *Let $\circledast$ and $\oplus$ be two binary operations in Σ. We write $t \downarrow_{\circledast,\oplus} u$ if, and only, if there are some t' and u' such that $t \rightsquigarrow t'$, $u \rightsquigarrow u'$, and $t' = u'$ can be proved by possibly using one application of axiom*

$$(x \oplus y) \circledast z = (x \circledast z) \oplus (y \circledast z)$$

at the top level—that is, either $t' \equiv u'$, $t' \equiv (t_1 \oplus t_2) \circledast t_3$ and $u' = (t_1 \circledast t_3) \oplus (t_2 \circledast t_3)$, or $t' \equiv (t_1 \circledast t_3) \oplus (t_2 \circledast t_3)$ and $u' \equiv (t_1 \oplus t_2) \circledast t_3$, for some t_1, t_2, t_3.

Lemma 4.3.17 *Let $\mathcal{T} = (\Sigma, \mathcal{L}, D)$ be a TSS. If $t \rightsquigarrow t'$ then $t \leftrightarrows t'$, for all $t, t' \in \mathbb{T}(\Sigma)$.*

Proof. By induction on the definition of $\rightsquigarrow$. The soundness of the rewrite rules

- $f(t,t) \leftrightsquigarrow t$, if $\mathcal{T}$ is in idempotence format with respect to f from [4], and
- $C[t] \rightsquigarrow C[t']$, if $t \rightsquigarrow t'$ and $\mathcal{T}$ is in a congruence format for $\leftrightarrows$,

is guaranteed by results in [4] and in the classic theory of structural operational semantics.

In order to check whether a rewriting rule preserves bisimilarity, in all cases apart from the first, the above definition relies on existing rule formats guaranteeing the validity of algebraic laws modulo bisimilarity, see [21], or on equations whose soundness with respect to bisimilarity is easy to check, such as

$$x +_l y = x \qquad \text{and} \qquad x +_r y = y.$$

This choice allows us to achieve an expressive and extensible rule format while retaining its syntactic nature. For instance, one may easily extend the rewriting relation $\rightsquigarrow$ with the following two clauses:

- $f(t_1, t_2) \leftrightsquigarrow f(t_2, t_1)$, if $\mathcal{T}$ is in the commutativity rule format with respect to f from [110], and
- $f(t, f(t', t'')) \leftrightsquigarrow f(f(t, t'), t'')$, if $\mathcal{T}$ is in the associativity rule format with respect to f from [49].

While proving the soundness of a left-distributivity law of the form

$$(x \oplus y) \circledast z \leftrightarrow (x \circledast z) \oplus (y \circledast z),$$

the validity of equivalences of the form

$$(t \oplus t') \circledast t'' = (t \circledast t'') \oplus (t' \circledast t'')$$

will be guaranteed by coinduction.

In Definition 4.3.18 to follow, which is the key ingredient in the definition of our second rule format for left distributivity, we shall use the relation $\downarrow_{\circledast,\oplus}$ to describe when a $\circledast$-defining rule d_1 is 'distributivity compliant' to a $\oplus$-defining rule d_2. The intuitive idea is that this will hold when those two rules can be combined to derive transitions from terms of the form $(p \oplus q) \circledast r$ and $(p \circledast r) \oplus (q \circledast r)$ that 'match' up to bisimilarity. Since the definition of distributivity compliance is quite technical, we find it useful to explain, by means of examples, the intuition behind it. For the sake of consistency and clarity, in the examples to follow, we shall use the same naming convention for substitutions that will be employed in Definition 4.3.18.

Suppose that the transition $(p \oplus q) \circledast r \xrightarrow{a} s$ is proved using rules d_1 and d_2, given below. Assume, furthermore, that

$$(d_1) \quad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y_1, y \xrightarrow{b} y_2\}}{x \circledast y \xrightarrow{a} t}$$

and that d_2 tests only one of its arguments, say

$$(d_2) \quad \frac{\{x \xrightarrow{a} x'\}}{x \oplus y \xrightarrow{a} t'}.$$

Then $s = \sigma_1(t)$, where

$$\begin{array}{rcl} \sigma_1 & = & [x \mapsto p \oplus q, y \mapsto r, x' \mapsto \sigma_2'(t'), y_1 \mapsto r_1, y_2 \mapsto r_2] \\ \sigma_2' & = & [x \mapsto p, y \mapsto q, x' \mapsto p'] \end{array}$$

and $p \xrightarrow{a} p'$, $r \xrightarrow{a} r_1$ and $r \xrightarrow{b} r_2$.

As highlighted by the proof of Theorem 4.3.6, rules d_2 and d_1 can be used to derive a transition $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} \sigma_2(t')$, where

$$\begin{array}{rcl} \sigma_2 & = & [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma_{1x}(t)] \\ \sigma_{1x} & = & [x \mapsto p, y \mapsto r, x' \mapsto p', y_1 \mapsto r_1, y_2 \mapsto r_2]. \end{array}$$

The transition $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} \sigma_2(t')$ will be deemed to 'match' $(p \oplus q) \circledast r \xrightarrow{a} s = \sigma_1(t)$ provided that

$$\sigma_1(t) \downarrow_{\circledast, \oplus} \sigma_2(t').$$

This will give a syntactically checkable guarantee that $\sigma_1(t) \leftrightarrows \sigma_2(t')$ holds.

Assume now that d_2 tests both its arguments, say

$$(d_2) \quad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} t'},$$

and that the transition $(p \oplus q) \circledast r \xrightarrow{a} s$ is proved using rule d_1 and rule d_2. Then $s = \sigma_1(t)$, where

$$\begin{array}{rcl} \sigma_1 & = & [x \mapsto p \oplus q, y \mapsto r, x' \mapsto \sigma_2'(t'), y_1 \mapsto r_1, y_2 \mapsto r_2] \\ \sigma_2' & = & [x \mapsto p, y \mapsto q, x' \mapsto p', y' \mapsto q'] \end{array}$$

and $p \xrightarrow{a} p'$, $q \xrightarrow{a} q'$, $r \xrightarrow{a} r_1$ and $r \xrightarrow{b} r_2$.

Let

$$(d_3) \quad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a}\!\!\!\!\rightarrow, y \xrightarrow{c} y'\}}{x \circledast y \xrightarrow{a} t''}.$$

Again, as highlighted by the proof of Theorem 4.3.6, rules d_2, d_1 and d_3 can be used to derive a transition $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} \sigma_{2x}(t')$, where

$$\begin{aligned} \sigma_{2x} &= [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma_{1x}(t), y' \mapsto \sigma'_{1y}(t'')] \\ \sigma'_{1y} &= [x \mapsto q, y \mapsto r, x' \mapsto q', y' \mapsto r'], \end{aligned}$$

and $p \circledast r \xrightarrow{a} \sigma_{1x}(t)$, $q \circledast r \xrightarrow{a} \sigma'_{1y}(t'')$, $q \xrightarrow{a} q'$ and $r \xrightarrow{c} r'$.

The transition $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} \sigma_{2x}(t')$ will be deemed to 'match' $(p \oplus q) \circledast r \xrightarrow{a} s = \sigma_1(t)$ provided that

$$\sigma_1(t) \downarrow_{\circledast,\oplus} \sigma_{2x}(t').$$

Again, this will give a syntactically checkable guarantee that $\sigma_1(t) \leftrightarrows \sigma_{2x}(t')$ holds. Note that, in this case, we also need to check this matching condition when the roles of rules d_1 and d_3 are swapped, since rule d_3 might be used to satisfy the x-testing premise of d_2 and rule d_1 might be used to satisfy the y-testing premise of that rule. In that case, our proof obligation is to show that

$$\sigma_1(t) \downarrow_{\circledast,\oplus} \sigma_{2y}(t'),$$

where

$$\begin{aligned} \sigma_{2y} &= [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma'_{1x}(t''), y' \mapsto \sigma_{1y}(t)] \\ \sigma'_{1x} &= [x \mapsto p, y \mapsto r, x' \mapsto p', y' \mapsto r'] \\ \sigma_{1y} &= [x \mapsto q, y \mapsto r, x' \mapsto q', y_1 \mapsto r_1, y_2 \mapsto r_2]. \end{aligned}$$

Definition 4.3.18 (Distributivity compliance up to $\rightsquigarrow$) *Let $\mathcal{T}$ be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of $\mathcal{T}$. Let d_1 be a $\circledast$-defining rule in $\mathcal{T}$ and d_2 be a $\oplus$-defining rule in $\mathcal{T}$. We say that d_1 is* distributivity compliant *to d_2 up to $\rightsquigarrow$, and we write it $d_1 \overset{\rightsquigarrow}{\sim} d_2$, whenever*

1. *rule d_1 is of the form (R1) and rule d_2 is of the form (R2),*
2. *the collection of positive y-testing premises in d_1 is of the form $\{y \xrightarrow{a_i} y_i \mid i \in I\}$, for some index set I, where all the variables are pairwise distinct, and*

3. *one of the following two cases applies:*

   (a) *d_2 has premises $\{x \xrightarrow{a} x'\}$ or $\{y \xrightarrow{a} y'\}$, and*

   $$\sigma_1(toc(d_1)) \downarrow_{\circledast,\oplus} \sigma_2(toc(d_2)),$$

   or

   (b) *d_2 has premises $\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}$ and, for each rule $d_3 \in D(\circledast, a)$,*

   - *the collection of positive y-testing premises in d_3 is of the form $\{y \xrightarrow{a_j} y_j \mid j \in J\}$, for some index set J, where all the variables are pairwise distinct,*
   - *$\sigma_1(toc(d_1)) \downarrow_{\circledast,\oplus} \sigma_{2x}(toc(d_2))$ and*
   - *$\sigma_1(toc(d_1)) \downarrow_{\circledast,\oplus} \sigma_{2y}(toc(d_2))$,*

   where the substitutions σ_1, σ_{1x}, σ_{1y}, σ_2, σ_{2x} and σ_{2y} are defined as follows, with p, q, p', q', r, r', and all the variables in $\{r_i \mid i \in I\} \cup \{r_j \mid j \in J\}$ being fresh and pairwise distinct variables.

   - $\sigma_1 = [x \mapsto p \oplus q, y \mapsto r, x' \mapsto \sigma_2'(toc(d_2)), y_i \mapsto r_i \ (i \in I)]$.
   - $\sigma_2 = [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma_{1x}(toc(d_1)), y' \mapsto \sigma_{1y}(toc(d_1))]$.
   - $\sigma_2' = [x \mapsto p, y \mapsto q, x' \mapsto p', y' \mapsto q']$.
   - $\sigma_{1x} = [x \mapsto p, y \mapsto r, x' \mapsto p', y_i \mapsto r_i \ (i \in I)]$.
   - $\sigma_{1x}' = [x \mapsto p, y \mapsto r, x' \mapsto p', y_j \mapsto r_j \ (j \in J)]$.
   - $\sigma_{1y} = [x \mapsto q, y \mapsto r, x' \mapsto q', y_i \mapsto r_i \ (i \in I)]$.
   - $\sigma_{1y}' = [x \mapsto q, y \mapsto r, x' \mapsto q', y_j \mapsto r_j \ (j \in J)]$.
   - $\sigma_{2x} = [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma_{1x}(toc(d_1)), y' \mapsto \sigma_{1y}'(toc(d_3))]$.
   - $\sigma_{2y} = [x \mapsto p \circledast r, y \mapsto q \circledast r, x' \mapsto \sigma_{1x}'(toc(d_3)), y' \mapsto \sigma_{1y}(toc(d_1))]$.

The reader should notice that, in order not to complicate the definition further by a more refined case distinction, in condition 3a of Definition 4.3.18, the substitution σ_2 is defined for both x' and y', even if in that case only one of them appears in rule d_2.

The following result is straightforward.

Theorem 4.3.19 (Decidability of $\overset{\sim}{\sim}$) *Let $\mathcal{T}$ be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of $\mathcal{T}$. Assume that the set of premises of each $\circledast$-defining rule*

is finite. Let d_1 be a $\circledast$-defining rule in $\mathcal{T}$ and d_2 be a $\oplus$-defining rule in $\mathcal{T}$. The problem of determining whether $d_1 \overset{\rightsquigarrow}{\sim} d_2$ holds is decidable.

Remark 4.3.20 *Note that $\overset{\rightsquigarrow}{\sim}$ performs only one rewriting step on both the terms. Clearly, extending Definition 4.3.18 in order to consider any finite amount of rewriting steps would not jeopardize Theorem 4.3.19.*

We now have all the necessary ingredients to define our second rule format for left distributivity.

Definition 4.3.21 (Second left-distributivity format) *A TSS $\mathcal{T}$ is in the* second left-distributivity format *for a binary operator $\circledast$ with respect to a binary operator $\oplus$ whenever, for each action a,*

1. *Fire$(\circledast, \oplus, a)$, and*
2. *if $D(\circledast, a) \neq \emptyset$ then $d_1 \overset{\rightsquigarrow}{\sim} d_2$, for each $d_1 \in D(\circledast, a)$ and for each $d_2 \in D(\oplus, a)$.*

We are now ready to formulate the two main theorems of this work.

Theorem 4.3.22 (Soundness of the second left-distributivity format) *Let $\mathcal{T}$ be a TSS. If $\mathcal{T}$ is in the second left-distributivity format for $\circledast$ with respect to $\oplus$ then*

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z).$$

Proof. A proof of this result may be found in Section 4.11.

Remark 4.3.23 *The above theorem holds true for any notion of distributivity compliance up to rewriting that is based on a rewriting relation $\rightsquigarrow$ over terms that has the following properties:*

- *$\rightsquigarrow \subseteq \leftrightarrows$ and*
- *$\rightsquigarrow$ is decidable.*

The latter requirement is not necessary for the soundness of the format. However, it is highly desirable from the point of view of applications. Indeed, in order to obtain a bona fide *rule format, the relation $\rightsquigarrow$ should be defined by using rules whose applicability can be checked syntactically, for instance using extant rule format for operational semantics. The proposal we presented in Definition 4.3.16 fits this requirement.*

The following result is straightforward, but important from the point of view of applications. In its statement, we use *Range*(f) to stand for the set of actions a for which there exists an a-emitting f-defining rule.

Theorem 4.3.24 (Decidability of the second rule format) *Let $\mathcal{T}$ be a TSS, and let $\circledast$ and $\oplus$ be two binary operators from the signature of $\mathcal{T}$. Assume that $Range(\circledast)$ is finite, that each $\circledast$-defining rule has a finite set of premises, and that $D(\circledast, a) \cup D(\oplus, a)$ is finite for each $a \in Range(\circledast)$. Then it is decidable whether $\mathcal{T}$ is in the second left-distributivity format for $\circledast$ with respect to $\oplus$.*

The import of Theorems 4.3.22 and 4.3.24 is that, when establishing that an operator $\circledast$ is left distributive with respect to an operator $\oplus$, it is sufficient to check whether the SOS specification for those operators meets the conditions of the format of Definition 4.3.21, which can be done effectively when the TSS under study is finitary.

4.4 Analyzing the distributivity compliance

In this section, we reduce the analysis of the distributivity-compliance relation $\rightsquigarrow$ to a syntactic check on the targets of the conclusions of the $\circledast$- and $\oplus$-defining rules. By analyzing different possible syntactic shapes for terms, we check which pairs of shapes can be related using the distributivity-compliance relation. This analysis is useful in order to avoid many of the substitutions involved in Definition 4.3.18, and, as witnessed by some of the examples in Section 4.5, to avoid all of them in many cases.

Table 4.1 summarizes our results. Even though the offered list is not exhaustive, which, at first sight, seems a challenging task to achieve, we believe Table 4.1 offers enough cases to avoid substitutions completely in most cases.

Table 4.1: Analysis of the distributivity-compliance pairs

	$\text{toc}(d_1)$	$\text{toc}(d_2)$	result	further requirements
1	$x' \circledast y$	x	$p \circledast r$	
2	$x' \circledast y$	y	$q \circledast r$	
3	x	$x' \oplus y'$	$p \oplus q$	$D(\circledast, a) = \{d_1\}$
4	x'	$x' \oplus y'$	$p' \oplus q'$	$D(\circledast, a) = \{d_1\}$
5	$x \circledast t$	$x' \oplus y'$	$(p \oplus q) \circledast \sigma(t)$	$D(\circledast, a) = \{d_1\}$, $x, x' \notin vars(t)$
6	$x' \circledast t$	$x' \oplus y'$	$(p' \oplus q') \circledast \sigma(t)$	$D(\circledast, a) = \{d_1\}$, $x, x' \notin vars(t)$
7	t	$x' \oplus y'$	$\sigma(t)$	$\oplus$ idempotent, $D(\circledast, a) = \{d_1\}$, $x, x' \notin vars(t)$
8	t	x'	$\sigma'(t)$	Condition 4 of Definition 4.3.7, $x \notin vars(t)$
9	t	y'	$\sigma'(t)$	Condition 4 of Definition 4.3.7, $x \notin vars(t)$

with $\sigma = [y \mapsto r, y_i \mapsto r_i \ (i \in I)]$ and $\sigma' = [y \mapsto r, x' \mapsto p', y_i \mapsto r_i \ (i \in I)]$

In Table 4.1, x and y are considered as the variables for the first and second argument, respectively, for both $\circledast$- and $\oplus$-defining rules. When the variable x' is mentioned, implicitly the considered rule has a premise $x \xrightarrow{a} x'$ (for a-emitting rules). Similarly, when the variable y' is mentioned, implicitly the rule considered has a premise $y \xrightarrow{a} y'$. The term t stands for a generic open term from the signature, and, following Definition 4.3.18, p, q and r are hypothetical closed terms applied to the distributivity equation in this way: $(p \oplus q) \circledast r \leftrightarrows (p \circledast r) \oplus (q \circledast r)$. The symbols p', q', and r_i, are considered as targets of possible transitions from p, q and r.

Table 4.1 is to be read as follows. First of all, $d_1 \in D(\circledast, a)$ and $d_2 \in D(\oplus, a)$, for some action a. In each row, the first column (column toc(d_1)) specifies the form of the target of the conclusion of the $\circledast$-defining rule d_1 (e.g., x in case of row 3), and the second column (column toc(d_2)) specifies the form of the target of the conclusion of the $\oplus$-defining rule d_2 (e.g., $x' \oplus y'$ in case of row 3). If the conditions in the column *further requirements* are satisfied (e.g., in row 3, d_1 is the only $\circledast$-defining and a-emitting rule), then the result of the transition of terms $(p \oplus q) \circledast r$ and $(p \circledast r) \oplus (q \circledast r)$ is specified by the term given in column *result* (e.g., $p \oplus q$ in row 3). In rows 5–6, the stated result is up to one application of the left-distributivity equation (1). The requirement $\oplus$ *idempotent* means that the operator $\oplus$ can be proved idempotent, e.g., by means of the rule format offered in [4].

The reader may want to notice that the first rule format of Section 4.3.2 is partly based on the analysis which leads to rows 8 and 9 in Table 4.1.

Theorem 4.4.1 (Soundness of Table 4.1) *Let $\mathcal{T}$ be a TSS. Let $\circledast$ and $\oplus$ be binary operations in the signature of $\mathcal{T}$ satisfying*

1. *Fire($\circledast, \oplus, a$), and*
2. *if $D(\circledast, a) \neq \emptyset$ then for each $d_1 \in D(\circledast, a)$ and for each $d_2 \in D(\oplus, a)$, the rules d_1 and d_2 match a row in Table 4.1.*

It holds that:

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z).$$

Proof. The proof of the theorem goes by a straightforward check of the conditions of Definition 4.3.18 on the combination specified in each row. For example, we discuss the case of row 7 in some detail below.

Applying the substitutions, we can see that on the left side of the distributivity equation $(p \oplus q) \circledast r \leftrightarrows (p \circledast r) \oplus (q \circledast r)$, we can prove the transition $(p \oplus q) \circledast r \xrightarrow{a} v$,

with $v = t[x \mapsto p \oplus q,\ y \mapsto r,\ x' \mapsto (x' \oplus y')[x \mapsto p,\ y \mapsto q,\ x' \mapsto p',\ y' \mapsto q'],\ y_i \mapsto r_i\ (i \in I)]$, and thus

$$v = t[x \mapsto p \oplus q, y \mapsto r, x' \mapsto p' \oplus q', y_i \mapsto r_i\ (i \in I)].$$

On the right side of the distributivity equation, we can prove the transition $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} v'$, with $v' = (x' \oplus y')[x \mapsto p \circledast r,\ y \mapsto q \circledast r,\ x' \mapsto t[x \mapsto p,\ y \mapsto r,\ x' \mapsto p',\ y_i \mapsto r_i\ (i \in I)]),\ y' \mapsto t[x \mapsto q,\ y \mapsto r,\ x' \mapsto q',\ y_i \mapsto r_i\ (i \in I)]$, and thus $v' = v'_1 \oplus v'_2$, where

$$\begin{array}{lcll} v'_1 & = & t[x \mapsto p, y \mapsto r, x' \mapsto p', y_i \mapsto r_i\ (i \in I)] & \text{and} \\ v'_2 & = & t[x \mapsto q, y \mapsto r, x' \mapsto q', y_i \mapsto r_i\ (i \in I)]. & \end{array}$$

From the column *further requirements* of row 7, we know that the variables x and x' do not appear in t, leading the two terms to be $v = t[y \mapsto r, y_i \mapsto r_i\ (i \in I)]$ and $v' = v \oplus v$. Since, as a further requirement, the operator $\oplus$ is idempotent with respect to bisimilarity, i.e., $x \oplus x \leftrightarroweq x$, we can conclude that

$$v' \downarrow_{\circledast,\oplus} v = t[y \mapsto r, y_i \mapsto r_i\ (i \in I)],$$

where $t[y \mapsto r, y_i \mapsto r_i\ (i \in I)]$ is the term stated in the column *result* of row 7.

4.5 Examples

In what follows, we apply the rule format provided in Section 4.3.3 in order to check some examples of left-distributivity laws whose validity cannot be inferred using Theorem 4.3.8.

Example 4.5.1 (Interleaving parallel composition and left choice) *As we remarked in Example 4.3.10, the equality*

$$(x +_l y) \parallel z \quad \leftrightarroweq \quad (x \parallel z) +_l (y \parallel z)$$

is sound. However, its soundness cannot be shown using Theorem 4.3.8, since the parallel composition operator $\parallel$ does not satisfy condition 2 in Definition 4.3.7. Indeed, x occurs in the target of the conclusion of the second rule for $\parallel$.

On the other hand, the validity of the above law can be shown by applying the rule format from Definition 4.3.21. Indeed, we observe that

- *the targets of the conclusions of the pair of rules*

$$(par_0)\ \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad (lc_0)\ \frac{x \xrightarrow{a} x'}{x +_l y \xrightarrow{a} x'},$$

  when instantiated as required in Definition 4.3.18, both become $p' \parallel r$, and

- *the targets of the conclusions of the pair of rules*

$$(par_1)\ \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad (lc_1)\ \frac{x \xrightarrow{a} x'}{x +_l y \xrightarrow{a} x'},$$

  when instantiated as required in Definition 4.3.18, become $(p +_l q) \parallel r'$ and $p \parallel r'$, with $(p +_l q) \parallel r' \rightsquigarrow p \parallel r'$.

Example 4.5.2 (Unit-delay operator and the choice operator from ATP) *Consider any TSS $\mathcal{T}$ containing the unit-delay operator $\lfloor\ \rfloor$ and the choice operator $+^*$ from ATP [112][2] and for which the transition relation $\xrightarrow{\chi}$ is deterministic. (The distinguished symbol χ denotes the passage of one unit of time.) The semantics of those operators is defined by the following rules, where $a \neq \chi$.*

$$(ud_a)\ \frac{x \xrightarrow{a} x'}{\lfloor x \rfloor(y) \xrightarrow{a} x'} \qquad (ud_\chi)\ \frac{}{\lfloor x \rfloor(y) \xrightarrow{\chi} y}$$

$$(extChl_a)\ \frac{x \xrightarrow{a} x'}{x +^* y \xrightarrow{a} x'} \qquad (extChr_a)\ \frac{y \xrightarrow{a} y'}{x +^* y \xrightarrow{a} y'}$$

$$(extTime)\ \frac{x \xrightarrow{\chi} x' \quad y \xrightarrow{\chi} y'}{x +^* y \xrightarrow{\chi} x' +^* y'}$$

We claim that $\mathcal{T}$ is in the second left-distributivity format for $\lfloor\ \rfloor$ with respect to $+^*$. Indeed, we observe that

- *the targets of the conclusions of the pair of rules $(ud_a, extChl_a)$ when instantiated as required in Definition 4.3.18, both become p',*

- *the targets of the conclusions of the pair of rules $(ud_a, extChr_a)$ when instantiated as required in Definition 4.3.18, both become q', and*

[2] In [112], the symbol of this operator is $\oplus$, whose use we prefer to avoid in this chapter for the sake of clarity.

- *the targets of the conclusions of the pair of rules $(ud_\chi, extTime)$ when instantiated as required in Definition 4.3.18, become r and $r +^* r$, with $r +^* r \rightsquigarrow r$ because $\mathcal{T}$ is in idempotence format with respect to $+^*$, as argued in [4, Example 9].*

The well-known law

$$\lfloor x +^* y \rfloor(z) \leftrightarrows \lfloor x \rfloor(z) +^* \lfloor y \rfloor(z)$$

thus follows from Theorem 4.3.22.

Table 4.1 can be used to match the targets of the conclusions as follows: the combination of ud_a and $extChl_a$ follows from row 8, the combination of ud_a and $extChr_a$ follows from row 9, and finally the combination of ud_χ and extTime follows from row 7.

Example 4.5.3 (Timed left merge and the choice operator from ATP) *Consider the TSS for ATP with the timed extension of the left-merge operator from Example 4.3.10 specified by the following rules, where $a \neq \chi$:*

$$(merge_a)\frac{x \xrightarrow{a} x'}{x \mathbin{\underline{\|}} y \xrightarrow{a} x' \parallel y} \qquad (merge_\chi)\frac{x \xrightarrow{\chi} x' \quad y \xrightarrow{\chi} y'}{x \mathbin{\underline{\|}} y \xrightarrow{\chi} x' \mathbin{\underline{\|}} y'}.$$

We claim that this TSS is in the second left-distributivity format for $\underline{\|}$ with respect to $+^*$. We limit ourselves to checking that the targets of the conclusions of the second rule for $\underline{\|}$ and rule extTime match when instantiated as required in Definition 4.3.18. This follows because, in all cases, the resulting terms yield an instance of the equality

$$(p' +^* q') \mathbin{\underline{\|}} r' = (p' \mathbin{\underline{\|}} r') +^* (q' \mathbin{\underline{\|}} r').$$

The law

$$(x +^* y) \mathbin{\underline{\|}} z = (x \mathbin{\underline{\|}} z) +^* (y \mathbin{\underline{\|}} z)$$

thus follows from Theorem 4.3.22.

Checking the conditions of the second rule format can be simplified by using the syntactic checks of Table 4.1, as follows: the combination $merge_a$, $extChl_a$ follows from row 8, the combination $merge_a$, $extChr_a$ follows from row 9 and the combination $merge_\chi$, extTime follows from row 6.

4.6 Examples of left-distributivity laws involving unary operators

In this section we apply the rule formats from Section 4.3 in order to prove left-distributivity laws involving unary operators from the literature. In order to do so, we turn unary operators into binary operators that simply ignore their right argument.

We begin with three examples that can be dealt with using Theorem 4.3.8.

Example 4.6.1 (Encapsulation and choice) *Consider the classic unary encapsulation operators ∂_H from ACP [24], where $H \subseteq \mathcal{L}$, with rules*

$$\frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \quad a \notin H.$$

It is well known that

$$\partial_H(x + y) \leftrightarrows \partial_H(x) + \partial_H(y), \tag{4.2}$$

where $+$ is the choice operator from Example 4.2.4.

We shall now argue that the validity of this equation can be shown using Theorem 4.3.8. To this end, we turn the encapsulation operators into binary operators that ignore their second argument. The above rules therefore become

$$\frac{x \xrightarrow{a} x'}{\partial_H(x, y) \xrightarrow{a} \partial_H(x', y)} \quad a \notin H.$$

Note that the rules for ∂_H and $+$ are in the first rule format for left distributivity from Definition 4.3.7. In particular, $\mathrm{Fire}(\partial_H, +, a)$ holds for each action a, because if there is an a-emitting rule for ∂_H then there is also an a-emitting rule for $+$. (Note that the converse only holds if $H = \emptyset$. This explains the asymmetric nature of the constraint $\mathrm{Fire}(\circledast, \oplus, a)$.) Therefore Theorem 4.3.8 yields the validity of the left-distributivity law

$$\partial_H(x + y, z) \leftrightarrows \partial_H(x, z) + \partial_H(y, z),$$

from which the soundness of (4.2) follows immediately.

Example 4.6.2 (Match operator and choice) *Consider the unary match operators* $[a = b]$ *from the* π*-calculus [132]*[3]*, where* $a, b \in \mathcal{L}$*, with rules*

$$\frac{x \xrightarrow{c} x'}{[a = b](x) \xrightarrow{c} x'} \quad \text{if } a = b,$$

where $c \in \mathcal{L}$*.*

It is well known that

$$[a = b](x + y) \leftrightarrows [a = b](x) + [a = b](y), \tag{4.3}$$

where $+$ *is the choice operator from Example 4.2.4.*

We shall now argue that the validity of this equation can be shown using Theorem 4.3.8. To this end, as above, we turn the match operators into binary operators that ignore their second argument. The above rules therefore become

$$\frac{x \xrightarrow{c} x'}{[a = b](x, y) \xrightarrow{c} x'} \quad \text{if } a = b.$$

Note that the rules for $[a = b]$ *and* $+$ *are in the first rule format for left distributivity from Definition 4.3.7. Therefore Theorem 4.3.8 yields the validity of the left-distributivity law*

$$[a = b](x + y, z) \leftrightarrows [a = b](x, z) + [a = b](y, z),$$

from which the soundness of (4.3) follows immediately.

Example 4.6.3 (Projection operator and choice) *Consider the unary projection operators* π_n *from ACP [24, 37], where* $n \geq 0$*, with rules*

$$\frac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')} \quad a \in \mathcal{L}.$$

It is well known that

$$\pi_n(x + y) \leftrightarrows \pi_n(x) + \pi_n(y), \tag{4.4}$$

where $+$ *is the choice operator from Example 4.2.4.*

We shall now argue that the validity of this equation can be shown using Theorem 4.3.8. Again, we turn the projection operators into binary operators that ignore their second

[3] Note that in the π-calculus a and b in the formula $[a = b]p$ are names and *not* labels.

argument. The above rules therefore become

$$\frac{x \xrightarrow{a} x'}{\pi_{n+1}(x,y) \xrightarrow{a} \pi_n(x',y)} \quad a \in \mathcal{L}.$$

Note that the rules for π_n and $+$ are in the first rule format for left distributivity from Definition 4.3.7. Therefore Theorem 4.3.8 yields the validity of the left-distributivity law

$$\pi_n(x+y,z) \leftrightarrow \pi_n(x,z) + \pi_n(y,z),$$

from which the soundness of (4.4) follows immediately.

Example 4.6.4 (Prefix operator and synchronous parallel operator) *Consider any TSS $\mathcal{T}$ containing the synchronous parallel operator $\|_{\mathcal{L}}$ from Example 4.3.11 and containing the following binary version of the prefix operator from CCS [96], where a ranges over a set of actions $\mathcal{L}$:*

$$(pref_a) = \frac{}{a.(x,y) \xrightarrow{a} x}.$$

We claim that $\mathcal{T}$ is in the second left-distributivity format for the prefix operator with respect to $\|_{\mathcal{L}}$. Let us pick an action a. Then the targets of the conclusions of $pref_a$ and of

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \|_{\mathcal{L}} y \xrightarrow{a} x' \|_{\mathcal{L}} y'},$$

which is the only a-emitting rule for $\|_{\mathcal{L}}$, both yield the term $p \|_{\mathcal{L}} q$ when instantiated as required in Definition 4.3.18. Therefore, Theorem 4.3.22 yields the validity of the law

$$a.(x \|_{\mathcal{L}} y, z) \leftrightarrow a.(x,z) \|_{\mathcal{L}} a.(y,z).$$

Turning the prefix operator back to its unary version, we obtain the soundness of the following equality:

$$a.(x \|_{\mathcal{L}} y) \leftrightarrow a.x \|_{\mathcal{L}} a.y.$$

Row 3 in Table 4.1 can be used to match the targets of the conclusions of the synchronous parallel composition and the prefix operators.

Example 4.6.5 (Unit-delay operator and choice operator) *Consider any TSS $\mathcal{T}$ that includes the choice operator $+^*$ from Example 4.5.2 and the following binary versions of the unit-delay operator:*

$$(delay_1) = \frac{}{(1)(x,y) \xrightarrow{\chi} x}.$$

We claim that $\mathcal{T}$ is in the second left-distributivity format for (1) with respect to $+^*$. To see this, it suffices to observe that the targets of the conclusions of the χ-emitting rules for those two operators, when instantiated as required in Definition 4.3.18, both yield the term $p +^* q$. Therefore, Theorem 4.3.22 yields the validity of the law

$$(1)(x +^* y, z) \leftrightarrows (1)(x,z) +^* (1)(y,z).$$

Turning the unit-delay operator back to its unary version, we obtain the well-known law

$$(1)(x +^* y) \leftrightarrows (1)(x) +^* (1)(y).$$

Row 3 in Table 4.1 can be used to match the targets of the conclusions of the delay rules for the unit-delay and choice operators.

4.7 Impossibility results

In this section we provide some impossibility results concerning the validity of the left-distributivity law. Unlike previous results about rule formats for algebraic properties, such as those surveyed in [21], we offer theorems to recognize when the left-distributivity law is guaranteed *not* to hold. When designing operational specifications for operators that are intended to satisfy a left-distributivity law, a language designer might also benefit from considering these kinds of negative results.

4.7.1 Left-inheriting operators

Our first negative result will concern a kind of left-inheriting operator, which we call strong left-inheriting and we now proceed to define.

Definition 4.7.1 (Forwarder operators) *Let* $\overrightarrow{k} = (k_1, k_2, \ldots, k_\ell)$*, where* $1 \leq \ell \leq n$ *and* $1 \leq k_1 < k_2 < \ldots < k_\ell \leq n$*. An operator* f *of arity* n *is a* $\overrightarrow{k}$-forwarder *if the following conditions hold for each action* a *and for all closed terms* $p_1, \ldots, p_n$*:*

- *if* $f(p_1 \ldots, p_{k_1}, \ldots, p_{k_2}, \ldots, p_{k_\ell}, \ldots, p_n) \xrightarrow{a}$ *then there is some* $1 \leq i \leq \ell$ *such that* $p_{k_i} \xrightarrow{a}$ *and*
- *for each* $1 \leq i \leq \ell$*, if* $p_{k_i} \xrightarrow{a}$ *then* $f(p_1 \ldots, p_{k_1}, \ldots, p_{k_2}, \ldots, p_{k_\ell}, \ldots, p_n) \xrightarrow{a}$.

Syntactic conditions to guarantee that an operator is a $\overrightarrow{k}$-forwarder can be given. However, this is beyond the scope of the present work.

Example 4.7.2 *As the reader can easily check, the left-merge operator* $\between$ *from Example 4.3.10 and the replication operator* ! *given by the rule below*

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \parallel !x} \quad (a \in \mathcal{L}),$$

where $\parallel$ *is the interleaving parallel composition operator from Example 4.3.10, are* (1)*-forwarders. On the other hand, the interleaving parallel composition operator and the choice operator* $+$ *from Example 4.2.4 are* $(1,2)$*-forwarders.*

Definition 4.7.3 (Forwarder contexts) *The grammar for* forwarder contexts for a variable x *is*

$$F[x] ::= x \mid f(x_1, \ldots, x_{i-1}, F[x], x_{i+1}, \ldots, x_n),$$

where f *is an* n*-ary operator,* $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ *are variables,* $F[x]$ *appears as the* i*th argument of* f*, and* f *is* $\overrightarrow{k}$*-forwarder with* i *appearing in* $\overrightarrow{k}$*.*

Lemma 4.7.4 *Assume that* $F[x]$ *is a forwarder context for a variable* x*. Then, for each closed substitution* σ *and for each action* a*, the following statements hold:*

1. *if* $\sigma(x) \xrightarrow{a}$ *then* $\sigma(F[x]) \xrightarrow{a}$*;*
2. *if* $\sigma(F[x]) \xrightarrow{a}$ *then there is some* $y \in vars(F[x])$ *such that* $\sigma(y) \xrightarrow{a}$*.*

Proof. Both claims can be shown by structural induction on $F[x]$.

Definition 4.7.5 (Strong left-inheriting operators) *Given a TSS* $\mathcal{T}$*, let* $\circledast$ *be a binary operator from the signature of* $\mathcal{T}$*. We say that* $\circledast$ *is* strong left-inheriting with respect to an action *a whenever each a-emitting* $\circledast$*-defining rule d has the form*

$$\frac{\Phi_x \cup \Phi_y}{x \circledast y \xrightarrow{a} F[x]},$$

where

- Φ_x *and* Φ_y *are sets of* x*-testing and* y*-testing formulae, respectively, whose subsets of positive premises are finite,*
- *no two formulae in* $\Phi_x \cup \Phi_y$ *contradict each other,*
- *each positive formula in* $\Phi_x \cup \Phi_y$ *has the form* $z \xrightarrow{b} z'$ *for some action* b *and variable* z'*,*
- *the variables* x*,* y *and the targets of the positive formulae in* $\Phi_x \cup \Phi_y$ *are all distinct, and*
- $F[x]$ *is a forwarder context for* x *with* $vars(F[x]) \subseteq vars(\Phi_x \cup \Phi_y) \cup \{x\}$*.*

Intuitively, not only does a strong left-inheriting operator inherit its left argument; it also makes sure that the inherited term may affect the next step of computation.

Theorem 4.7.6 (Impossibility Theorem: strong left-inheriting operators) *Given a TSS* $\mathcal{T}$*, let* $\circledast$ *be a binary operator in the signature of* $\mathcal{T}$*. Assume that*

- *the set of actions is infinite,*
- *the signature of* $\mathcal{T}$ *contains the inaction constant from Remark 4.3.9, the prefix operators from CCS (see Example 4.6.4) and the choice operator from Example 4.2.4,*
- $\circledast$ *is a strong left-inheriting operator with respect to some action* $a \in \mathcal{L}$*, and*
- *there is some* a*-emitting and* $\circledast$*-defining rule.*

Then

$$(x + y) \circledast z \leftrightarrows (x \circledast z) + (y \circledast z).$$

The proof of Theorem 4.7.6, which may be found in Section 4.12, relies on the fact that, when $(p + q) \circledast r \xrightarrow{a} s_1$ for some action a and closed terms p, q, r and s_1, the term s_1 has both the initial capabilities of p and q because s_1 has some occurrence of the term $p + q$ in a forwarder context, and $+$ is itself a $(1, 2)$-forwarder. On the other hand, if $(p \circledast r) + (q \circledast r) \xrightarrow{a} s_2$, for some s_2, then s_2 is never able to have both of the initial capabilities of p and q simultaneously, since $+$ performs a choice.

Using Theorem 4.7.6, we obtain, for instance, that:

- $(x + y) \parallel z \leftrightarrows (x \parallel z) + (y \parallel z)$
- $a.(x + y) \leftrightarrows (a.x) + (a.y)$

- $!(x + y) \underline{\leftrightarrow} (!x) + (!y)$

For the last two cases, in order to apply the above-mentioned theorem, one needs to consider the binary version of the action prefixing operator from Example 4.6.4 and the binary version of the replication operator, which ignores its second argument and can be defined along the lines we followed in the examples in Section 4.6.

4.7.2 The use of negative premises

We now present two results that rely on the use of negative premises in rules.

Definition 4.7.7 (Always Moving Operators) *Given a TSS $\mathcal{T}$, we say that an operator f from the signature of $\mathcal{T}$ with arity n is* always moving for action *a whenever $f(\overrightarrow{p}) \xrightarrow{a}$, for each n-tuple of closed terms $\overrightarrow{p}$.*

For example, an n-ary operator f, with $n \geq 1$, is always moving for action a when the set of rules $D(f, a)$ contains

- either some rule d with $\mathrm{hyps}(d) = \emptyset$,
- or rules d_1, d_2 with $\mathrm{hyps}(d_1) = \{x_1 \xrightarrow{a} x_1'\}$ and $\mathrm{hyps}(d_2) = \{x_1 \overset{a}{\not\to}\}$.

An example of operator that is always moving for action a is the prefixing operator $a.\_$.

Remark 4.7.8 *It is possible to find syntactic conditions on the set of rules for some operator f guaranteeing that f is always moving. For instance, the decidable logic of initial transition formulae offered in [7], which is able to reason about firability of GSOS rules, can be used in order to check whether operators are always moving. The development of rule formats for always-moving operators is, however, orthogonal to the gist of this work and therefore we do not address it here.*

Theorem 4.7.9 *Given a TSS $\mathcal{T}$, let $\circledast$ and $\oplus$ be binary operators in the signature of $\mathcal{T}$. Assume that*

1. *the signature of $\mathcal{T}$ contains at least one constant,*
2. *$a \in \mathcal{L}$,*
3. *$\circledast$ is always moving for action a, and*
4. *the set of premises of each a-emitting and $\oplus$-defining rule contains either $x \overset{a}{\not\to}$ or $y \overset{a}{\not\to}$.*

Then

$$(x \oplus y) \circledast z \underline{\leftrightarrow} (x \circledast z) \oplus (y \circledast z),$$

and any triple of closed terms witnesses the above inequivalence.

Proof. Let $\mathcal{T}$ be a *TSS*, and let $\circledast$ and $\oplus$ be binary operators of the signature of $\mathcal{T}$. Let p, q and r be arbitrary closed terms, which exist since the signature of $\mathcal{T}$ contains at least one constant.

Since $\circledast$ is always moving for action a, we have that $(p \oplus q) \circledast r \xrightarrow{a}$, $(p \circledast r) \xrightarrow{a}$ and $(q \circledast r) \xrightarrow{a}$. As each a-emitting and $\oplus$-defining rule d is, by assumption, such that $x \overset{a}{\nrightarrow} \in \text{hyps}(d)$ or $y \overset{a}{\nrightarrow} \in \text{hyps}(d)$, none of those rules can be used to prove an a-labelled transition for $(p \circledast r) \oplus (q \circledast r)$. It follows that

$$(p \oplus q) \circledast r \underline{\leftrightarrow} (p \circledast r) \oplus (q \circledast r),$$

as required.

In what follows we offer a result that ensures the invalidity of the distributivity law when negative premises appear in $\circledast$-defining rules.

Theorem 4.7.10 *Let $\mathcal{T}$ be a TSS whose signature contains a binary operator $\circledast$, the inaction constant* **0***, the prefix operators from CCS and the choice operator. Assume that there is some action a such that the only a-emitting $\circledast$-defining rule in $\mathcal{T}$ has the form*

$$(d) \quad \frac{\Phi_x \cup \Phi_y}{x \circledast y \xrightarrow{a} t},$$

where

- *Φ_x and Φ_y are sets of x-testing and y-testing formulae, respectively, whose subsets of positive premises are finite,*
- *no two formulae in $\Phi_x \cup \Phi_y$ contradict each other,*
- *each positive formula in $\Phi_x \cup \Phi_y$ has the form $z \xrightarrow{b} z'$ for some action b and variable z',*
- *the variables x, y and the targets of the positive formulae in $\Phi_x \cup \Phi_y$ are all distinct, and*
- *$\{x \overset{b}{\nrightarrow} \mid b \in L\} \subseteq \Phi_x$, for some non-empty set of actions L.*

Then

$$(x + y) \circledast z \underline{\leftrightarrow} (x \circledast z) + (y \circledast z).$$

Proof. Let $\{x \xrightarrow{a_i} x_i \mid i \in I\}$ and $\{y \xrightarrow{b_j} y_j \mid j \in J\}$, where I and J are finite index sets, be the collections of positive premises in Φ_x and Φ_y, respectively. Define

$$\begin{aligned} p &= \sum_{i \in I} a_i.\mathbf{0} \quad \text{and} \\ r &= \sum_{j \in J} b_j.\mathbf{0}. \end{aligned}$$

By the assumption of the theorem, the closed substitution σ mapping x to p, y to r and all the other variables to $\mathbf{0}$ satisfies the premises of d. Therefore, we have that

$$p \circledast r \xrightarrow{a} \sigma(t).$$

Let $q = b.\mathbf{0}$ for some $b \in L$. Then,

$$(p \circledast r) + (q \circledast r) \xrightarrow{a} \sigma(t).$$

On the other hand, the term $(p + q) \circledast r$ does not afford an a-labelled transition because $p + q \xrightarrow{b} \mathbf{0}$ and therefore no closed substitution mapping x to $p + q$ can satisfy the premises of d, which is the only a-emitting $\circledast$-defining rule in $\mathcal{T}$. This means that

$$(p + q) \circledast r \underline{\leftrightarrow} (p \circledast r) + (q \circledast r),$$

and the claim follows.

Example 4.7.11 *Let $>$ be an irreflexive partial order over $\mathcal{L}$. The priority operator Θ from [27] is specified by the following rules:*

$$\frac{x \xrightarrow{a} x', \quad x \overset{b}{\nrightarrow} \ (\forall b > a)}{\Theta(x) \xrightarrow{a} \Theta(x')} \quad (a \in \mathcal{L}).$$

The binary version of that operator can be defined following the lines presented in the examples in Section 4.6. Theorem 4.7.10, when applied to the binary version of Θ, yields the well-known fact that, when $>$ is a non-trivial partial order,

$$\Theta(x + y) \underline{\leftrightarrow} \Theta(x) + \Theta(y).$$

Indeed, if $>$ is non-trivial, then there are actions a and b with $a < b$. The single a-emitting rule for the binary version of Θ has a negative premise of the form $x \overset{b}{\nrightarrow}$, and therefore Theorem 4.7.10 is applicable to derive the above inequivalence.

4.8 Conclusions

In this work we have provided two rule formats guaranteeing that certain binary operators are left distributive with respect to choice-like operators. As witnessed by the wealth of examples we discussed in the main body of this study, the rule formats are general enough to cover relevant examples from the literature. In particular, they can also be applied to establish the validity of left-distributivity laws involving unary operators. This can be achieved by simply considering unary operators as binary operators that ignore their second argument.

We have also offered conditions that allow one to recognize the invalidity of the left-distributivity law in the context of left-inheriting operators and in the presence of negative premises. Such conditions can be applied to well-known examples of *invalid* left-distributivity laws.

By a straightforward adaptation of our formats, one can obtain syntactic conditions on SOS rules guaranteeing right distributivity.

The research presented in this article opens several interesting lines for future investigation. First of all, our rule formats can be easily adapted to obtain rule formats guaranteeing the validity of right-distributivity laws of the form

$$x \circledast (y \oplus z) = (x \circledast y) \oplus (x \circledast z).$$

The rule formats we have presented should also be extended in order to handle examples of distributivity laws where $\oplus$ is not 'choice-like'. It would also be interesting to see whether one can relax the syntactic constraints of the rule formats presented in this chapter substantially, while preserving their soundness and ease of application.

Last, but not least, we intend to find further 'impossibility theorems' along the lines of those we presented in Section 4.7.

This future work will lead to a better understanding of the semantic nature of distributivity properties and of its links to the syntax of SOS rules.

4.9 Proof of Theorem 4.3.6

Instead of proving Theorem 4.3.6 we prove a stronger theorem. In what follows, when we say $(p \oplus q) \circledast r \xrightarrow{a}$ using rules d_1 and d_2, the considered transition is provable by the $\circledast$-defining rule d_1, possibly using the $\oplus$-defining rule d_2 to prove a transition $(p \oplus q) \xrightarrow{a} p'$ satisfying the set $\Phi_x(d_1)$ of x-testing premises in d_1. We say $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a}$ using rules d_2, d_1 and d_3, with the straightforward analogous meaning, using d_1 to prove a transition from $(p \circledast r)$ satisfying $\Phi_x(d_2)$ and d_3 to prove a transition from $(q \circledast r)$ satisfying $\Phi_y(d_2)$.

Theorem 4.9.1 *Let T be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of T. Suppose that Fire$(\circledast, \oplus, a)$, for some actions a. Then, for all closed terms p, q, and r,*

- *if $(p \oplus q) \circledast r \xrightarrow{a}$ using rules d_1 and d_2 then $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a}$ using rules d_2, d_1 and d_1.*
- *$(p \circledast r) \oplus (q \circledast r) \xrightarrow{a}$ using rules d_2, d_1 and d_3 then $(p \oplus q) \circledast r \xrightarrow{a}$ using rules d_1 or d_3, and d_2.*

It is easy to see that Theorem 4.9.1 implies Theorem 4.3.6.

Theorem 4.9.1 can be proved along the lines of Theorem 4.3.8 and we therefore omit the details.

4.10 Proof of Theorem 4.3.8

Let T be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of T. Assume that the rules for $\circledast$ and $\oplus$ are in the first rule format for left distributivity. We show the following two claims, where p, q, r, s are arbitrary closed terms and a is any action:

1. If $(p \oplus q) \circledast r \xrightarrow{a} s$ then $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$.
2. If $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$ then $(p \oplus q) \circledast r \xrightarrow{a} s$.

We consider each of the above claims in turn.

1. Assume that $(p \oplus q) \circledast r \xrightarrow{a} s$. We shall prove that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$.

Since $(p \oplus q) \circledast r \xrightarrow{a} s$ and $\mathrm{Fire}(\circledast, \oplus, a)$ holds, there are a rule d_1 of the form

$$\frac{(\emptyset \text{ or } \{x \xrightarrow{a} x'\}) \ \cup \ \Phi_y}{x \circledast y \xrightarrow{a} t}$$

and a closed substitution σ such that

- $\sigma(x) = p \oplus q$,
- $\sigma(y) = r$,
- $\sigma(t) = s$ and
- σ satisfies the premises of d_1.

We shall argue that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$ by considering two cases, depending on whether d_1 has a premise of the form $x \xrightarrow{a} x'$.

(a) Case: d_1 has no x-testing premise. In this case, rule d_1 can be used to infer that $p \circledast r \xrightarrow{a} s$ and $q \circledast r \xrightarrow{a} s$ both hold. Indeed, recall that $x \notin \mathit{vars}(\Phi_y)$ by the constraints of the rule form $(R1)$ and $x \notin \mathit{vars}(t)$ by constraint 2 in Definition 4.3.7. Therefore, the closed substitution $\sigma[x \mapsto p]$ satisfies the premises of d_1 and is such that

$$\sigma[x \mapsto p](x \circledast y \xrightarrow{a} t) = p \circledast r \xrightarrow{a} s.$$

A similar reasoning using the closed substitution $\sigma[x \mapsto q]$ shows that $q \circledast r \xrightarrow{a} s$ is also provable using d_1 as claimed. The first and third condition in Definition 4.3.3 yield the existence of some rule $d_2 \in D(\oplus, a)$ of the form

$$\frac{(\{x \xrightarrow{a} x'\} \text{ or } \{y \xrightarrow{a} y'\} \text{ or } \{x \xrightarrow{a} x', y \xrightarrow{a} y'\})}{x \oplus y \xrightarrow{a} t}.$$

By constraint 3 of Definition 4.3.7, d_2 has a target variable of one of its premises as target of its conclusion. Therefore, regardless of the set of premises of d_2, we can instantiate that rule using any closed substitution mapping x to $p \circledast r$, y to $q \circledast r$ and both x' and y' to s to infer that

$$(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s,$$

as required.

(b) Case: d_1 has a premise of the form $x \xrightarrow{a} x'$. In this case, as σ satisfies the premises of d_1, we have that

$$\sigma(x) = p \oplus q \xrightarrow{a} \sigma(x').$$

The above transition can be proved using a rule $d_2 \in D(\oplus, a)$ of the form

$$\frac{(\{x \xrightarrow{a} x'\} \text{ or } \{y \xrightarrow{a} y'\} \text{ or } \{x \xrightarrow{a} x', y \xrightarrow{a} y'\})}{x \oplus y \xrightarrow{a} t'},$$

where, by constraint 3, $t' = x'$ or $t' = y'$. Assume, without loss of generality, that $t' = y'$. Then $y \xrightarrow{a} y'$ is a premise of rule d_2 and

$$q \xrightarrow{a} \sigma(x').$$

So, instantiating rule d_1 above using $\sigma[x \mapsto q]$, we have that

$$\sigma[x \mapsto q](x \circledast y) = q \circledast r \xrightarrow{a} \sigma[x \mapsto q](t) = \sigma(t) = s.$$

(Recall that $x \notin vars(t)$ by constraint 2 in Definition 4.3.7.) If d_2 does not have any x-testing premise then the above transition can be used to satisfy its premise and we can infer

$$(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s,$$

as required. Assume now that d_2 has $x \xrightarrow{a} x'$ as a premise, and therefore has the form

$$\frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} y'}.$$

Since the transition $p \oplus q \xrightarrow{a} \sigma(x')$ is proved using d_2, there is some p' such that $p \xrightarrow{a} p'$. Recall that, by the assumptions for this case of the proof,

$$d_1 = \frac{\{x \xrightarrow{a} x'\} \cup \Phi_y}{x \circledast y \xrightarrow{a} t}.$$

Then the substitution $\sigma[x \mapsto p, x' \mapsto p']$ satisfies the premises of d_1, and we can deduce that

$$\sigma[x \mapsto p, x' \mapsto p'](x \circledast y) = p \circledast r \xrightarrow{a} \sigma[x \mapsto p, x' \mapsto p'](t) = \sigma[x' \mapsto p'](t).$$

Using rule d_2 and any substitution that maps x to $p \circledast r$, x' to $\sigma[x' \mapsto p'](t)$, y to $q \circledast r$ and y' to s, we may conclude that

$$(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s,$$

as required.

2. Assume that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$. We shall prove that $(p \oplus q) \circledast r \xrightarrow{a} s$.

Since $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s$ and $\mathrm{Fire}(\circledast, \oplus, a)$ holds, there are a rule d_2 of the form

$$\frac{(\{x \xrightarrow{a} x'\} \text{ or } \{y \xrightarrow{a} y'\} \text{ or } \{x \xrightarrow{a} x', y \xrightarrow{a} y'\})}{x \oplus y \xrightarrow{a} t},$$

where, by constraint 3, $t = x'$ or $t = y'$, and a closed substitution σ such that

- $\sigma(x) = p \circledast r$,
- $\sigma(y) = q \circledast r$,
- $\sigma(t) = s$ and
- σ satisfies the premises of d_2.

Assume, without loss of generality, that $t = x'$. Therefore $x \xrightarrow{a} x'$ is a premise of d_2 and

$$\sigma(x) = p \circledast r \xrightarrow{a} s = \sigma(x').$$

Since $p \circledast r \xrightarrow{a} s$, there are some rule

$$d_1 = \frac{(\emptyset \text{ or } \{x \xrightarrow{a} x'\}) \ \cup \ \Phi_y}{x \circledast y \xrightarrow{a} t'}$$

and a closed substitution σ' such that

- $\sigma'(x) = p$,
- $\sigma'(y) = r$,
- $\sigma'(t') = s$ and
- σ' satisfies the premises of d_1.

We shall argue that $(p \oplus q) \circledast r \xrightarrow{a} s$ by considering two cases, depending on whether d_1 has a premise of the form $x \xrightarrow{a} x'$.

(a) Case: d_1 has no x-testing premise.

Consider the substitution $\sigma'[x \mapsto p \oplus q]$. Since $x \notin vars(\Phi_y)$ and σ' satisfies the premises of d_1, it follows that $\sigma'[x \mapsto p \oplus q]$ also satisfies Φ_y. Therefore, we can instantiate rule d_1 with $\sigma'[x \mapsto p \oplus q]$ to infer that

$$\sigma'[x \mapsto p \oplus q](x \circledast y) = (p \oplus q) \circledast r \xrightarrow{a} \sigma'[x \mapsto p \oplus q](t') = \sigma'(t') = s,$$

as required. (Recall that $\circledast$ is non-left-inheriting by condition 2 in Definition 4.3.7.)

(b) Case: d_1 has a premise of the form $x \xrightarrow{a} x'$. Then,

$$d_1 = \frac{\{x \xrightarrow{a} x'\} \cup \Phi_y}{x \circledast y \xrightarrow{a} t'}.$$

As σ' satisfies the premises of d_1, we have that

$$\sigma'(x) = p \xrightarrow{a} \sigma'(x').$$

If $x \xrightarrow{a} x'$ is the only premise of rule d_2, then we can use that rule and the above transition to infer that

$$p \oplus q \xrightarrow{a} \sigma'(x').$$

Consider now the closed substitution $\sigma'[x \mapsto p \oplus q]$. This substitution satisfies the premises of rule d_1, because so does σ' and $x \notin vars(\Phi_y)$. Therefore, instantiating rule d_1 with $\sigma'[x \mapsto p \oplus q]$, we may derive the transition

$$(p \oplus q) \circledast r \xrightarrow{a} \sigma'[x \mapsto p \oplus q](t') = \sigma'(t') = s,$$

as required.

Assume now that $x \xrightarrow{a} x'$ is *not* the only premise of rule d_2. Then, because of the assumptions of this case,

$$d_2 = \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} x'}.$$

Recall that we used the above rule and the closed substitution σ to prove the transition

$$(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} s.$$

Therefore we have that

$$\sigma(y) = q \circledast r \xrightarrow{a} \sigma(y').$$

Using condition 4 in Definition 4.3.7 and the form of the rules d_1 and d_2, this means that there are a rule

$$d_3 = \frac{\{x \xrightarrow{a} x'\} \cup \Phi'_y}{x \circledast y \xrightarrow{a} t''}$$

and a closed substitution $\hat{\sigma}$ such that

- $\hat{\sigma}(x) = q \xrightarrow{a} \hat{\sigma}(x')$,
- $\hat{\sigma}(y) = r$,
- $\hat{\sigma}(t'') = \sigma(y')$ and
- $\hat{\sigma}$ satisfies Φ'_y.

Using rule d_2 with premises $p \xrightarrow{a} \sigma'(x')$ and $q \xrightarrow{a} \hat{\sigma}(x')$, we obtain that

$$p \oplus q \xrightarrow{a} \sigma'(x').$$

Finally, instantiating rule d_1 with the closed substitution $\sigma'[x \mapsto p \oplus q]$, we infer the transition

$$\sigma'[x \mapsto p \oplus q](x \circledast y) = (p \oplus q) \circledast r \xrightarrow{a} \sigma'[x \mapsto p \oplus q](t') = \sigma'(t') = s,$$

as required.

This completes the proof.

4.11 Proof of Theorem 4.3.22

Let $T = (\Sigma, \mathcal{L}, D)$ be a TSS. Assume that T is in the second left-distributivity format for $\circledast$ with respect to $\oplus$. We shall prove that

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z).$$

To this end, it suffices to show that the relation

$$\mathcal{R} = \{((p \oplus q) \circledast r, (p \circledast r) \oplus (q \circledast r)) \mid p, q, r \in \mathbb{C}(\Sigma)\} \cup \leftrightarrows$$

is a bisimulation.

Let us pick an action a and closed terms p, q and r. We now prove the following two claims:

1. If $(p \oplus q) \circledast r \xrightarrow{a} v_1$ then $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} v_2$, for some v_2 such that $v_1 \mathcal{R} v_2$.
2. If $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} v_2$ then $(p \oplus q) \circledast r \xrightarrow{a} v_1$, for some v_1 such that $v_1 \mathcal{R} v_2$.

We consider these two claims separately.

1. Assume that $(p \oplus q) \circledast r \xrightarrow{a} v_1$ for some closed term v_1. This means that $(p \oplus q) \circledast r \xrightarrow{a} v_1$ using rules d_1 and d_2, for some $\circledast$-defining rule d_1 and some $\oplus$-defining rule d_2.

   By Theorem 4.9.1, $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} v_2$, for some closed term v_2, using rules d_2, d_1 and d_1. We shall now show that $v_1 \mathcal{R} v_2$.

   As T is in the second left-distributivity format for $\circledast$ with respect to $\oplus$, we have that $d_1 \overset{\rightsquigarrow}{\sim} d_2$. We distinguish two cases depending on whether the set of premises of d_2 is a singleton.

   - Case: $\mathrm{hyps}(d_2) = \{x \xrightarrow{a} x'\}$ or $\mathrm{hyps}(d_2) = \{y \xrightarrow{a} y'\}$. In both of the cases, the term v_1 is formed by exactly the substitutions of condition 3a in Definition 4.3.18, when the variable p' is used as a term such that $p \xrightarrow{a} p'$, similarly q' for q, and each r_i for y_i, $i \in I$. Thus, $v_1 = \sigma_1(\mathrm{toc}(d_1))$ and, for the same reasons, $v_2 = \sigma_2(\mathrm{toc}(d_2))$. Now, by the definition of $\overset{\rightsquigarrow}{\sim}$, we have that $v_1 \rightsquigarrow v_1'$ and $v_2 \rightsquigarrow v_2'$, for some v_1' and v_2' with $v_1' = v_2'$, by possibly using one application of axiom

     $$(x \oplus y) \circledast z = (x \circledast z) \oplus (y \circledast z)$$

     at the top level. Since $v_1 \leftrightarrows v_1'$ and $v_2 \leftrightarrows v_2'$ hold by Lemma 4.3.17, by possibly using the transitivity of bisimilarity, we may conclude that $v_1 \mathcal{R} v_2$, as required.

   - Case: $\mathrm{hyps}(d_2) = \{x \xrightarrow{a} x', y \xrightarrow{a} y'\}$. In this case, by condition 3b in Definition 4.3.18, the bisimilarity proven in the previous case is guaranteed

for all the possible pairs of $\circledast$-defining rules, and this also includes the case when the two premises of rule d_2 are both satisfied using rule d_1.

2. Assume that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} v_2$ for some closed term v_2. This transition can be proved using rules d_2, d_1, d_3, for some $\oplus$-defining rule d_2 and some $\circledast$-defining rules d_1 and d_3.

   By Theorem 4.9.1, $(p \oplus q) \circledast r \xrightarrow{a} v_1$, for some closed term v_1, using rules d_1 or d_3 and d_2. We now argue that $v_1 \mathcal{R} v_2$. By condition 3b in Definition 4.3.18, reasoning as above, $v_1 \mathcal{R} v_2$ is guaranteed for all the possible pairs of $\circledast$-defining rules, including the case when the transition $(p \oplus q) \circledast r \xrightarrow{a} v_1$ is proved using d_1 and d_2 or using d_3 and d_2.

This completes the proof.

4.12 Proof of Theorem 4.7.6

Let T be a *TSS* and let $\circledast$ be a binary operator of the signature of T. Assume the hypotheses of Theorem 4.7.6.

Let us pick an a-emitting and $\circledast$-defining rule d. By the hypotheses of the theorem, d has the form

$$\frac{\Phi_x \cup \Phi_y}{x \circledast y \xrightarrow{a} F[x]},$$

where

- Φ_x and Φ_y are sets of x-testing and y-testing formulae, respectively, whose subsets of positive premises are finite,
- no two formulae in $\Phi_x \cup \Phi_y$ contradict each other,
- each positive formula in $\Phi_x \cup \Phi_y$ has the form $z \xrightarrow{b} z'$ for some action b and variable z',
- the variables x, y and the targets of the positive formulae in $\Phi_x \cup \Phi_y$ are all distinct, and
- $F[x]$ a forwarder context for x with $vars(F[x]) \subseteq vars(\Phi_x \cup \Phi_y) \cup \{x\}$.

Since the signature of T contains the inaction, the prefix operators and the choice operator, and no two formulae in $\Phi_x \cup \Phi_y$ contradict each other, it is easy to construct three terms p, q, and r such that

1. p 'satisfies' Φ_x,
2. if $x \overset{b}{\nrightarrow} \in \Phi_x$, then $q \overset{b}{\nrightarrow}$,
3. r 'satisfies' Φ_y,
4. $p \overset{b}{\rightarrow}$, $q \overset{b}{\nrightarrow}$ and $r \overset{b}{\nrightarrow}$, for some action b,
5. $q \overset{c}{\rightarrow}$, $p \overset{c}{\nrightarrow}$ and $r \overset{c}{\nrightarrow}$, for some action c, and
6. the depth of p and r is one—that is, for all action b and c, and closed terms p' and r', if $p \overset{b}{\rightarrow} p'$ then $p' \overset{c}{\nrightarrow}$, and if $r \overset{b}{\rightarrow} r'$ then $r' \overset{c}{\nrightarrow}$.

Conditions 4 and 5 can be met because, by assumption, the set of actions is infinite.

We claim that

$$(p + q) \circledast r \underline{\leftrightarrow} (p \circledast r) + (q \circledast r).$$

To see this, observe that, since $+$ is a $(1, 2)$-forwarder operator, due to conditions 1 and 2, $p + q$ 'satisfies' Φ_x. By condition 3, the rule d fires with some closed substitution σ mapping x to $p + q$ and y to r. Thus $(p + q) \circledast r \overset{a}{\rightarrow} \sigma(F[x])$. By conditions 4–5 and Lemma 4.7.4, we have that $\sigma(F[x]) \overset{b}{\rightarrow}$ and $\sigma(F[x]) \overset{c}{\rightarrow}$.

Assume now that $(p \circledast r) + (q \circledast r) \overset{a}{\rightarrow} s$, for some s. We will now argue that $\sigma(F[x]) \underline{\leftrightarrow} v$, proving our claim that

$$(p + q) \circledast r \underline{\leftrightarrow} (p \circledast r) + (q \circledast r).$$

Indeed, suppose that $p \circledast r \overset{a}{\rightarrow} s$. Since $\circledast$ is strong left-inheriting with respect to an action a, we have that there are an a-emitting $\circledast$-defining rule of the form

$$\frac{\Phi'_x \cup \Phi'_y}{x \circledast y \overset{a}{\rightarrow} F'[x]},$$

satisfying the requirements in Definition 4.7.5 and a closed substitution σ' such that $s = \sigma'(F'[x])$. By conditions 5 and 6, using Lemma 4.7.4 we have that $s \overset{c}{\nrightarrow}$. Therefore $\sigma(F[x]) \underline{\leftrightarrow} s$.

If $q \circledast r \overset{a}{\rightarrow} s$ then, reasoning in similar fashion using conditions 4 and 6 as well as Lemma 4.7.4, we infer that $s \overset{b}{\nrightarrow}$. Therefore $\sigma(F[x]) \underline{\leftrightarrow} s$, and we are done.

Chapter 5

Structural Operational Semantics with Binders

When evil men burn and bomb, good men must build and bind.
Martin Luther King, Jr.

5.1 Introduction

The development of a formal semantics for programming and specification languages is a necessary first step towards rigorous reasoning about them. For instance, a formal semantics allows one to prove the correctness of language implementations, and is a prerequisite for proving the validity of program optimizations. Operational semantics is a widely-used methodology to define formal semantics for computer languages, which represents the execution of programs as step-by-step development of an abstract machine. *Structural Operational Semantics* (SOS) was introduced by Gordon Plotkin in [119], reprinted in [65], as a logical and structural approach to defining operational semantics. The logical structure of SOS specifications supports a variety of reasoning principles that can be used to prove properties of programs whose semantics is given using SOS. Moreover, SOS language specifications can be used for rapid prototyping of language designs and to provide experimental implementations of computer languages.

Thanks to its intuitive appeal and flexibility, SOS has become the de facto standard for defining operational semantics, and a wealth of programming and executable

specification languages have been given formal semantics using it. In recent years much work on the underlying theory as well as on the practice of SOS has been carried out—see, e.g., [18, 109] and [45, 71, 108], respectively. However, a substantial amount of work remains to be done in this rapidly-evolving area of research. This chapter will focus on one of the crucial aspects in the definition of semantic models for programming and specification languages that has so far received relatively little attention in the literature of the meta-theory of SOS, i.e., the treatment of concepts such as variables, names and binders.

Many programming and specification languages make use of the concepts of names and binders. For example, in the π-calculus [97, 98, 132], names are first-class objects and the whole language is built on the idea that concurrent agents communicate by exchanging names. Binders are syntactic constructs that are used to scope the use of names in expressions. Examples of binders are input-prefixing operators, recursion combinators, restriction operators, infinite-sum operators and the time-integration operator [29, 96, 67, 132].

In this chapter we propose a formal framework for the handling of names in SOS, called *Nominal SOS*, which is based on the nominal techniques of Gabbay, Pitts, and Urban [61, 145].

In the semantics of most nominal calculi some basic notions such as α-conversion and substitution are used. We show that these notions can be naturally captured in the Nominal SOS framework. Moreover, we specify two of the most prominent examples of nominal calculi, namely the lazy λ-calculus and the early π-calculus, in Nominal SOS and show that our specifications coincide operationally with the original definitions of [2] and [132], respectively. Finally, we define a notion of nominal bisimilarity naturally arising from our framework. We show that in the case of the π-calculus our notion coincides with the well-known open bisimilarity, [132, 131]. On the other hand, we prove that nominal bisimilarity is not a satisfactory notion of equivalence over the lazy λ-calculus. However, one of the most interesting notions of bisimilarity in the context of the λ-calculus is the *applicative bisimilarity* due to Samson Abramsky, [2]. We define this notion of equivalence in the framework of Nominal SOS and prove that coincides with its original counter-part.

The reader must bear in mind that this chapter should be considered as containing the basic developments of the framework of Nominal SOS, accompanied by some examples of its application. Refinements and extensions of the framework are possible and left as future work. The investigation of the theory of Nominal SOS

is also part of our future research plans. The reader may find in Section 5.9 an outline of our plans for future research. The overarching aim is to develop the framework of Nominal SOS in a way that is comparable to that of the standard theory of SOS, as surveyed in, e.g., [18, 109], and to hopefully establish Nominal SOS as a framework of reference for the study and the development of the theory of languages with first-class notions of names and binders.

Nominal SOS is not the only approach studied so far in the literature that aims at a uniform treatment of binders and names in the operational semantics of programming and specification languages. We are aware of a number of existing approaches that accommodate variables and binders inside variations on the SOS framework, and we discuss the most relevant approaches in Section 5.8.

Structure of the chapter. The rest of this chapter is organized as follows. In Section 5.2 we define nominal terms as used in the rest of the chapter. In Section 5.3 we define Nominal SOS, an SOS framework extended with names and binders. We show in Section 5.4 how α-conversion and different types of substitutions can be accommodated in the Nominal SOS framework. In Section 5.5, we give the definitions of the λ- and the π-calculus in our framework and show their correspondence with the original presentations. In Section 5.6, we define the notion of nominal bisimilarity and show that coincides with open bisimilarity over the early π-calculus. In Section 5.7, we formulate the notion of applicative bisimilarity in the framework of Nominal SOS and show that it coincides with the original notion of Abramsky. Section 5.8 discusses related works in some detail and Section 5.9 concludes the chapter by pointing out some directions for future work. For the sake of readability, proofs of some results and some technical definitions are collected in a series of sections that follow Section 5.9.

5.2 Nominal terms

The following definitions of *sorts* and *nominal signature* are familiar from [145].

Definition 5.2.1 (Sorts) *Sorts are defined inductively by the following grammar:*

$$\sigma ::= \mathbf{1} \mid \delta \mid \mathbb{A} \mid [\mathbb{A}]\sigma \mid \sigma \times \sigma,$$

where $\mathbf{1}$ *is the unit sort,* δ *is a base sort,* $\mathbb{A}$ *is an atom sort, and* $\times$ *denotes pairing.*

In the above-given grammar $[\mathbb{A}]\sigma$ denotes an abstraction sort. Intuitively, $[\mathbb{A}]\sigma$ is a sort whose elements are functions from objects of sort $\mathbb{A}$ to objects of sort σ. As is standard, pair sorts will associate to the left, so that $\sigma_1 \times \sigma_2 \times \sigma_3$ stands for $(\sigma_1 \times \sigma_2) \times \sigma_3$.

Definition 5.2.2 (Nominal Signature) *A* nominal signature Σ *is a triple* (Δ, A, F), *where*

1. Δ *is a set of* base sorts *ranged over by* δ,
2. A *is a set of* atom sorts *ranged over by* $\mathbb{A}$, *and*
3. F *is a set of* operators $f_{(\sigma_1 \times \ldots \times \sigma_n) \to \delta}$, *denoting a function symbol* f *with* arity $(\sigma_1 \times \ldots \times \sigma_n) \to \delta$, *where* $n \geq 0$.

A function symbol of arity $(\sigma_1 \times \sigma_2 \times \sigma_3) \to \delta$ can be applied to three arguments of sorts σ_1, σ_2 and σ_3, respectively, and the term resulting from this application is of sort δ.

For each atom sort $\mathbb{A}$, we fix a countably infinite set of *atoms* $a_{\mathbb{A}}$, $b_{\mathbb{A}}$, $c_{\mathbb{A}}$, $d_{\mathbb{A}}$, $n_{\mathbb{A}}$, $m_{\mathbb{A}}$ and, for each sort σ, we assume a countably infinite set V_σ of *variable symbols* x_σ, y_σ, z_σ.

We sometimes write just f, a, b, c, d, n, m, and x, y, z, leaving arities and sorts implicit (but still present). We assume that all these sets of symbols are pairwise disjoint.

Definition 5.2.3 (Nominal Terms) *Given a signature* $\Sigma = (\Delta, A, F)$, *the set of* nominal terms *over the signature* Σ *is denoted by* $\mathbb{T}(\Sigma)$ *and it is defined as follows, where we write* t_σ *for a term* t *of sort* σ*:*

$$t \quad ::= \quad x_\sigma \mid a_{\mathbb{A}} \mid ([a_{\mathbb{A}}]t_\sigma)_{[\mathbb{A}]\sigma} \mid (f_{(\sigma_1 \times \ldots \times \sigma_n) \to \delta}(t_{\sigma_1}, \ldots, t_{\sigma_n}))_\delta$$

where $\mathbb{A} \in A$, $a_{\mathbb{A}} \in \mathbb{A}$, $x_\sigma \in V_\sigma$ *and* f *is a function symbol in* F *with arity* $(\sigma_1 \times \ldots \times \sigma_n) \to \delta$.

When Σ is understood or irrelevant, we may write just $\mathbb{T}$ in lieu of $\mathbb{T}(\Sigma)$. The subscripts of nominal terms control sorting and we tend to omit them when they are clear from the context or immaterial. We call $[a]t$ an *abstraction* ('of a in t').

We wish to clarify the role of atoms and variables in Nominal SOS. As in the ordinary theory of SOS, we treat variables x, y, z, ..., as meta-variables, see [18] and [109], that range over the terms of the language, according to their sort.

On the other hand, atoms are named elements that a user can employ in programs. We call them atoms for historical reasons and, as stated in [145] with efficacy,

> [...] partly to indicate that the internal structure of such names is irrelevant to us: all we care about is their identity (i.e. whether or not one atom is the same as another) and that the supply of atoms is inexhaustible.

These named elements may serve various purposes. The most typical and also prominent example of usage of atoms is that, like formal parameters in procedures or function definitions, they represent placeholders for terms *yet to come*, in a parameter passing fashion. This is the case, for instance, of the λ-calculus, [33]. For the sake of concreteness, we give below a possible nominal signature for λ-terms.

The nominal syntax of the λ-calculus is constructed using a base sort L for λ-terms, an atom sort A and the following function symbols.

1. $\lambda(\_) : [A]L \to L$: A unary function symbol for embedding abstractions inside terms;
2. $\_\ \_ : (L \times L) \to L$: A binary function symbol for application.

The correspondence between nominal λ-terms and those of the λ-calculus is straightforward, and will be formalized later in the chapter. For example, the nominal term $(\lambda([a](a\ a))\ \ \lambda([a](a\ a)))$ represents the λ-term $(\lambda a.(a\ a)\ \ \lambda a.(a\ a))$. In this case, atoms are employed in order to model the variables of the object language that is being formalized.

The reader will see in the next section that the user can define how programs deal with occurrences of atoms by means of deduction rules similar to those of ordinary SOS.

Variables $x, y, z, \ldots$ act instead on a meta-level with respect to the object language. The reader that is familiar with ordinary SOS will notice that in the following deduction rule

$$\frac{}{\lambda([a]x) \to \lambda([a]x)}$$

the variable x ranges over all the terms over the signature described above. For instance, the rule above can be applied in order to prove transitions $\lambda([a]a) \to \lambda([a]a)$,

when x is mapped to a, and $\lambda([a]\lambda([b]b)) \to \lambda([a]\lambda([b]b))$, when x is mapped to $\lambda([b]b)$.

For a nominal term t, the following definitions will be useful in the remainder of the chapter.

- $vars(t)$ denotes the set of variables that occur in t.
- $\mathcal{A}(t)$ stands for the set of atoms that occur in t. For example, $\mathcal{A}(\lambda([a](a\ b))) = \{a, b\}$.
- $ba(t)$ is the set of atoms a for which there exists a subterm $[a]t'$ in t, i.e., the set of abstracted atoms in t. For example, $ba(\lambda([a](a\ b))) = \{a\}$.
- $fa(t)$ is the set of atoms a in $\mathcal{A}(t)$ that have an occurrence in t that is not within the scope of an abstraction $[a]t'$, for some term t'. We call $fa(t)$ the set of *free* atoms of t. For example, $fa(\lambda([a](a\ b))) = \{b\}$ and also $fa(\lambda([a]a)\ \ a) = \{a\}$.
- We say that an atom a is *fresh* in t whenever $a \notin fa(t)$. We also say that a term t is *binding-closed*[1] if $fa(t) = \emptyset$, i.e., the term t does not contain free atoms.

These sets will play a role in some proofs to follow. Their formal definitions are thus provided in Section 5.10.

We say that a nominal term is *closed* if it contains no variables. It is called *open* otherwise. For example, a and $[a]f(b)$ are closed terms, but x and $[a]y$ are open terms. Note that neither a nor $[a]f(b)$ is binding-closed. The set of closed terms in $\mathbb{T}(\Sigma)$ is denoted by $\mathbb{C}(\Sigma)$ and, again, where Σ is understood or irrelevant, we may write just $\mathbb{C}$. The sets of binding-closed terms in $\mathbb{T}(\Sigma)$ and those in $\mathbb{C}(\Sigma)$ are denoted by $\mathbb{T}(\Sigma)^0$ and $\mathbb{C}(\Sigma)^0$, respectively[2]. A *substitution* ρ over the signature Σ is a function of type $V \to \mathbb{T}(\Sigma)$[3]. We assume that substitutions are *sort-respecting*, i.e. that $\rho(x)$ and x have the same sort for each $x \in V$. We extend the domain of substitutions to terms homomorphically and write $t\rho$ for the result of applying the substitution ρ to the term t. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a *closed substitution*. That the substitutions must respect the sort of variables is

[1] Binding-closed terms corresponds to those that in literature are usually said *closed*. For instance, in the context of the λ-calculus the λ-term $\lambda a.\lambda b.(a\ b)$ is closed, as it does not contain free variables, see [2, 33]. We adopt a different nomenclature in order to avoid confusion with the standard concept of closed term of SOS, i.e., a term that contains no variables.

[2] The notation adopted for the set of binding-closed terms follows a standard notation. For instance the set of λ-terms is typically denoted by Λ and the set of closed λ-terms is typically denoted by Λ^0, see [2, 33].

[3] We use the symbol ρ for substitutions in place of the standard symbol σ in order to avoid the confusion that would otherwise arise with sorts.

a necessary requirement in the framework of Nominal SOS. Consider for instance the atom sort A, a base sort L and a binary function symbol f of arity $(A \times A) \rightarrow L$, i.e. the operator f accepts two atoms as arguments. A sort-respecting substitution ρ guarantees that in the term $f(x, y)\rho$, the variables x and y are mapped to atoms in A as expected, and not to some other type of terms.

5.3 Nominal SOS

Suppose a is an atom and t is a term of some sort. We call a formula $a\#t$ a *freshness assertion*. In what follows we give a derivation system in order to derive freshness assertions. Before embarking on the formal definition, we want to clarify the notion of freshness by means of the following examples, which use the nominal syntax for λ-terms defined in the previous section. The reader can consider the λ-terms defined in the previous section and notice the following facts.

- The atom a is fresh in $\lambda([b]c)$, as it does not appear in it. Thus the assertion $a\#\lambda([b]c)$ should be derivable.
- The atom a is *not* fresh in $\lambda([b]a)$, as it appears *free* in that term. Thus, the assertion $a\#\lambda([b]a)$ should not be derivable.
- The atom a *is* fresh in $\lambda([b]\lambda([a]c))$. In fact, the actual bound name in the abstraction $[a]c$ is considered immaterial and so it is hidden to an external observer. Thus, the assertion $a\#\lambda([b]\lambda([a]c))$ should be derivable, and so should $b\#\lambda([b]\lambda([a]c))$.

We now proceed to formalize the derivation of freshness assertions.

Definition 5.3.1 (Freshness derivation rules) *Let Σ be a nominal signature, and let the atom a and the term t be over the signature Σ. We write $\vdash a\#t$ when $a\#t$ may be derived using the following rules, where a and b are distinct atoms.*

$$\frac{}{a\#b} \qquad \frac{a\#t_1, \ldots, a\#t_n}{a\#f(t_1, \ldots, t_n)} \qquad \frac{}{a\#[a]t} \qquad \frac{a\#t}{a\#[b]t}$$

These derivation rules are familiar from existing work [145, 54]. As a matter of notation, we often simply write $a\#t$ for $\vdash a\#t$. The following theorem states the

correctness of the proof system for freshness assertions given above with respect to the freshness definition of the previous section.

Theorem 5.3.2 (Correctness of Freshness Derivations) *Let Σ be a nominal signature and let the atom a and the term t be over the signature Σ. It holds that a is fresh in t if and only if $a\#t$.*

A routine induction proves Theorem 5.3.2.

We are now ready to define the notion of *nominal transition system specification* whose rules employ the freshness assertions defined above.

Definition 5.3.3 (Nominal Transition System Specification) *A* nominal transition system specification *(*NTSS*) is a triple* (Σ, R, D) *consisting of:*

1. *A nominal signature Σ;*
2. *A set of* (transition) relation symbols *R. To each $r \in R$ we associate a* (transition relation) arity *which is a sort of the form $\sigma \times \sigma_l \times \sigma'$. We may call: σ the 'sort of the* source *of the transition', σ' the 'sort of the* target *of the transition', and σ_l the 'sort of the* label *of the transition'.*
3. *A set of derivation rules D (see below).*

Given an NTSS T, we denote with $\mathcal{A}(T)$ the set of atoms of the signature of T. For a relation $r \in R$ with arity $\sigma \times \sigma_l \times \sigma'$, if σ_l is the unit sort $\mathbf{1}$ then we say that *r has no label*. If σ' is also the unit sort, then r is a predicate symbol. We may silently drop σ_l (and σ') if they are the unit sort.

For a relation $r \in R$ with arity $\sigma \times \sigma_l \times \sigma'$, a *positive transition formula* is written $t \xrightarrow{l}_r t'$, where t is a possibly open term of sort σ (we call it the *source* term), l is a possibly open term of sort σ_l (we call it the *label*), and t' is a possibly open term of sort σ' (we call it the *target* term).

For the same relation r, we write $t \not\xrightarrow{l}_r$ for a *negative transition formula*, where t is of sort σ and l is of sort σ_l. A *transition formula* is a positive or negative transition formula.

A *derivation rule* is of the form

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\} \quad \{t_j \not\xrightarrow{l_j}_{r_j} \mid j \in J\} \quad \{a_k \# t_k \mid k \in K\}}{t \xrightarrow{l}_r t'}$$

where

- I, J and K are *indexing sets*,
- $\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\}$ is a set of positive transition formulae, called the *positive premises* of the rule,
- $\{t_j \xrightarrow{l_j}\!\!\!\!\!\!\rightarrow_{r_j} \mid j \in J\}$ is a set of negative transition formulae, called the *negative premises* of the rule,
- $\{a_k \# t_k \mid k \in K\}$ is a set of freshness assertions, called the *freshness premises* of the rule, and
- $t \xrightarrow{l}_r t'$ is a positive transition formula, which we call the *conclusion* of the rule.

We call t, l, and t' the *source*, the *label* and the *target* of the rule, respectively.

Substitutions are also extended to formulae, sets of formulae and rules in the natural way. For a derivation rule d and a substitution ρ, the rule $d\rho$ is called a *substitution instance* of d.

We call a derivation rule an *axiom* if I, J and K are empty. A derivation rule is *positive* when the index set J is empty. An NTSS is *positive* when all its deduction rules are positive.

Positive NTSS's are much easier to deal with than general ones and come with a natural notion of semantics, i.e., the set of provable transitions.

Given closed terms t and t', and a label l, the intended reading of $t \xrightarrow{l}_r t'$ is: t can make an r-transition with label l to t'. We write $t \xrightarrow{l}\!\!\!\!\!\!\rightarrow_r$ to mean that there is no term t' such that $t \xrightarrow{l}_r t'$. A positive NTSS gives these intuitions formal meaning using a notion of 'derivable transition', which we now define.

Definition 5.3.4 *Let T be an NTSS. The* derivable transitions *of T are inductively defined as follows. Suppose that*

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\} \quad \{a_k \# t_k \mid k \in K\}}{t \xrightarrow{l}_r t'}$$

is a rule in T, and suppose ρ is a closed substitution over the signature of T. If

- *$t_i\rho \xrightarrow{l_i\rho}_{r_i} t'_i\rho$ is derivable, for every $i \in I$, and*
- *$a_k \# t_k\rho$ is derivable (using the rules in Definition 5.3.1), for every $k \in K$,*

then $t\rho \xrightarrow{l\rho}_r t'\rho$ is derivable.

This means that the set of derivable transitions is the least collection of transitions that is 'closed under the application of the rules.'

5.3.1 Semantics of NTSS's

All the NTSSs we consider in this chapter are positive, and Definition 5.3.4 suffices to give their semantics.

To give a semantics to NTSS's in general, one has to define a meaning for negative transitions, i.e., when a negative transition is 'provable'. This has been a source of complications in the theory of SOS and several proposals for such a notion exist [147]. The most widely accepted notion of semantics for TSS's involving negative transitions is that of least three-valued stable model. To define this concept, we need two auxiliary definitions, namely provable transition rules and consistency, which are given below.

Definition 5.3.5 (Provable transition rules) *A closed deduction rule is called a* transition rule *when it is of the form* $\frac{N}{\phi}$ *with* N *a set of* negative formulae. *An NTSS* T proves $\frac{N}{\phi}$, *denoted by* $T \vdash \frac{N}{\phi}$, *when there is a well-founded upwardly branching tree with closed formulae as nodes and of which*

- *the root is labelled by* ϕ;
- *if a node is labelled by* ψ *and the labels of the nodes directly above it form the set* K *then:*
  - ψ *is a negative formula and* $\psi \in N$, *or*
  - ψ *is a positive formula and* $\frac{K}{\psi}$ *is a substitution instance of a deduction rule in* T.

Definition 5.3.6 (Contradiction and consistency) *The formula* $t \xrightarrow{l}_r t'$ *is said to* contradict $t \xnrightarrow{l}_r$, *and vice versa. For two sets* Φ *and* Ψ *of formulae,* Φ contradicts Ψ *when there is a* $\phi \in \Phi$ *that contradicts a* $\psi \in \Psi$. *We write* $\Phi \nvDash \Psi$, *and say that* Φ *is* consistent *with* Ψ, *when* Φ *does not contradict* Ψ.

It immediately follows from the above definition that contradiction and consistency are symmetric relations on (sets of) formulae. We now have all the necessary ingredients to define the semantics of NTSSs in terms of three-valued stable models.

Definition 5.3.7 (Three-valued stable model) *A pair* (C, U) *of disjoint sets of positive closed transition formulae is called a* three-valued stable model *for an TSS* T *when the following conditions hold:*

- *for each* $\phi \in C$*, there is a set* N *of negative formulae such that* $T \vdash \frac{N}{\phi}$ *and* $C \cup U \vDash N$*, and*
- *for each* $\phi \in U$*, there is a set* N *of negative formulae such that* $T \vdash \frac{N}{\phi}$ *and* $C \vDash N$*.*

C *stands for* Certainly *and* U *for* Unknown*; the third value is determined by the formulae not in* $C \cup U$*. The* least *three-valued stable model is a three-valued stable model that is the least one with respect to the ordering on pairs of sets of formulae defined as* $(C, U) \leq (C', U')$ *iff* $C \subseteq C'$ *and* $U' \subseteq U$*.*

In the literature [124, 147] (in the setting without names and binders), it has been shown that every TSS admits a least three-valued stable model with respect to the information theoretic ordering (i.e., $(C, U) \leq (C', U')$ when $C \subseteq C'$ and $U' \subseteq U$). This result easily extends also to our setting for NTSSs.

We say that an NTSS is *complete* when for its least three-valued stable model it holds that $U = \emptyset$. If an NTSS is complete, we write $p \xrightarrow{l}_r p'$ in lieu of $(p \xrightarrow{l}_r p') \in C$. All the NTSSs considered in this chapter are complete.

5.4 Substitution and α-conversion

Substitution and α-equivalence play a key role in the definition of the semantics of calculi with binders. We will now show how those notions can be accommodated within the framework of Nominal SOS.

Atoms inequality First of all, notice that it is possible to employ freshness premises in order to check inequality between two atoms. As a matter of fact, the premise $a\#b$ holds when a and b are different atoms. In the remainder of this chapter, some rules need to check inequality of atoms. In order to increase the readability of rules, and let the reader understand clearly the meaning behind some premises, we shall write sometimes a premise $a \neq t$ to mean $a\#t$, with t an atom or a variable of atom sort. The checking of atoms equality is never employed in the remainder of this chapter. As future extensions of the framework of Nominal SOS, we are planning to allow also for negative freshness premises and this type

of premises can be used to express equality of atoms, as an atom a is not fresh in an atom b only when a and b are the same atom.

5.4.1 Substitution transitions

Substitution is a natural notion required in the semantics of most nominal calculi. The well-known notion of substitution for these calculi is subtly different from the naive syntactic notion in that it should avoid "capture". For example, replacing b with a in $[a]b$, if done naively, results in $[a]a$, which captures the free name a by the abstraction surrounding it. Hence, a substitution is allowed only as long as the variable that is abstracted by the binder is fresh in the substituting term.

In Nominal SOS, we can model this freshness requirement within our framework (instead of in the meta-language as in most other approaches, such as [40]) by means of freshness premises, which are native to the nominal framework.

For a given nominal signature, the deduction rules for substitution transitions can be generated automatically. In what follows, we give the procedure to generate the deduction rules for the *term-for-atom* and *atom-for-atom* substitutions.

Term-for-atom substitutions are typically employed by higher-order calculi, such as the λ-calculus, CHOCS [140] and the Higher-Order π-calculus [129], just to mention a few. Given a nominal signature, we proceed to generate the following types of deduction rules with the goal of proving transitions of the form $t_1 \xrightarrow{a \overset{T}{\mapsto} t_2} t_3$ for some atom a and terms t_1, t_2 and t_3. This type of transition should be read as *the term t_2 replaces the atom a in the term t_1 leading to the term t_3*. More specifically, as in the framework of Nominal SOS labels are terms, given A and L as the sorts for atoms and terms of a signature, respectively, $\overset{T}{\mapsto}$ is a function symbol of arity $(A \times L) \to L$, i.e., an operator that accepts an atom and a term as argument and returns a term. The capitalized superscript T in the label of this transition is present to recall that this type of transition is the term-for-atom substitution. In the next paragraph, we model the atom-for-atom substitution and its corresponding label will have A in place of T.

In reading the following rules, the reader should remember that a premise $a \neq x$, where x is a variable ranging over the sort of atoms, means $a\#x$. For all atoms a and function symbols f, we have the following rules.

$$a \xrightarrow{a \overset{T}{\mapsto} z} z \ (\mathrm{a1_{Ts}}) \qquad \frac{a \neq x}{a \xrightarrow{x \overset{T}{\mapsto} z} a} \ (\mathrm{a2_{Ts}}) \qquad \frac{x \xrightarrow{y \overset{T}{\mapsto} z} x' \quad a\#z \quad a \neq y}{[a]x \xrightarrow{y \overset{T}{\mapsto} z} [a]x'} \ (\mathrm{abs1_{Ts}})$$

$$[a]x \xrightarrow{a \overset{T}{\mapsto} z} [a]x \ (\mathrm{abs2_{Ts}}) \qquad \frac{\left\{ x_i \xrightarrow{y \overset{T}{\mapsto} z} x_i' \mid 0 < i \leq n \right\}}{f(x_1, x_2, \ldots, x_n) \xrightarrow{y \overset{T}{\mapsto} z} f(x_1', x_2', \ldots, x_n')} \ (\mathrm{f_{Ts}})$$

The reader can infer the sort of the variables used in the rules by their usage. For instance, the variable x that occurs in rule ($\mathrm{a2_{Ts}}$) is of atom sort, since it is employed as the first argument of the substitution transition label. In the same rule, the variable z is of the sort of the terms.

Inequality of atoms is employed in rule ($a\mathrm{2_{Ts}}$) in order to model the fact that a substitution that aims at replacing the atom b with a term t is ineffective when applied to atoms that are different from b, i.e., $a \xrightarrow{b \overset{T}{\mapsto} t} a$ for all terms t and distinct atoms a and b. Inequality is also used in rule ($abs\mathrm{1_{Ts}}$) in order to model the fact that the substitution within an abstraction is effective only when it does not try to substitute the abstracted atom.

Term-for-atom substitution transitions are deterministic.

Lemma 5.4.1 (Determinism of substitution transitions.) *Let T be an NTSS containing the rules for term-for-atom substitution as defined above. For all closed terms t, t', t'' and w, and for each atom a, it holds that if $t \xrightarrow{a \overset{T}{\mapsto} w} t'$ and $t \xrightarrow{a \overset{T}{\mapsto} w} t''$ then $t' = t''$.*

A simple structural induction proves Lemma 5.4.1. To see this, it suffices only to notice that rules ($\mathrm{a1_{Ts}}$) and ($\mathrm{a2_{Ts}}$) cannot be applied simultaneously, and also ($\mathrm{abs1_{Ts}}$) and ($\mathrm{abs2_{Ts}}$) cannot be applied simultaneously.

The reader may be more familiar with the syntactic substitution operation, defined below, where M and N are closed terms and a and b are distinct atoms.

$$\begin{aligned}
a[N/a] &= N \\
a[N/b] &= a \\
([a]M)[N/a] &= [a]M \\
([a]M)[N/b] &= [a](M[N/b]) \ \text{ if } a \text{ is fresh in } N \\
f(M_1, M_2, \ldots, M_n)[N/a] &= f(M_1[N/a], M_2[N/a], \ldots, M_n[N/a])
\end{aligned}$$

The following theorem states that the two notions (substitution transitions and syntactic substitutions) correspond.

Theorem 5.4.2 (Correctness of Substitution Transitions) *Let T be an NTSS. Let M and N be closed terms, and a be an atom. Then, it holds that $M \xrightarrow{a \overset{T}{\mapsto} N} M'$ if and only if $M' = M[N/a]$.*

The reader can find the proof of Theorem 5.4.2 in Section 5.13.

Atom-for-atom substitution is used in calculi such as the π-calculus [132, 98] and its variants. The same set of rules provided for the term-for-atom substitution are able to model the atom-for-atom substitution. The rules are applied in order to generate transitions of the form $t_1 \xrightarrow{a \overset{A}{\mapsto} b} t_2$ for some atoms a and b and terms t_1 and t_2. More specifically, given A and L as the sorts for atoms and terms of a signature, respectively, $\overset{A}{\mapsto}$ is a function symbol of arity $(A \times A) \to L$, i.e., an operator that accepts two atoms as argument and returns a term. Since atom-for-atom substitution transitions will play a crucial role in the modelling of α-conversion presented later, we prefer to make these rules explicit and we show them below. In reading the following rules, the reader should remember that a premise $a \neq x$, where x is a variable ranging over the sort of atoms, means $a\#x$. In what follows, the variables y and z are of atom sort and a ranges over atoms.

$$a \xrightarrow{a \overset{A}{\mapsto} z} z \ (\mathrm{a1_{As}}) \qquad \frac{a \neq x}{a \xrightarrow{x \overset{A}{\mapsto} z} a} \ (\mathrm{a2_{As}}) \qquad \frac{x \xrightarrow{y \overset{A}{\mapsto} z} x' \quad a \neq z \quad a \neq y}{[a]x \xrightarrow{y \overset{A}{\mapsto} z} [a]x'} \ (\mathrm{abs1_{As}})$$

$$[a]x \xrightarrow{a \overset{A}{\mapsto} z} [a]x \ (\mathrm{abs2_{As}}) \qquad \frac{\left\{ x_i \xrightarrow{y \overset{A}{\mapsto} z} x_i' \mid 0 < i \leq n \right\}}{f(x_1, x_2, \ldots, x_n) \xrightarrow{y \overset{A}{\mapsto} z} f(x_1', x_2', \ldots, x_n')} \ (\mathrm{f_{As}})$$

As in the previous example, the reader can infer the sort of the variables used in the rules by their usage. For instance, differently from the previous case, the variable z that occur in the rule $(\mathrm{a2_{As}})$ is of atom sort.

It is important not to confuse the semantic substitutions ρ defined in Section 5.2 and the ones defined in this section. The former is on the meta-level of the semantics of Nominal SOS and acts on variables. It is a semantic meta-operation that is used in order to instanciate the rules of Nominal SOS and prove transitions. This notion, also, allows for capture of atoms. For instance, we can apply the

axiom (abs2$_{\text{As}}$) with a substitution ρ which maps x to a and z to b for some atoms a and b, in order to prove the transition $[a]a \xrightarrow{a \overset{A}{\mapsto} b} [a]a$.

The substitutions defined in this section are instead user-defined and their purpose is replacing atoms by terms, modelling thus the fact that atoms are variables of the object language and represent placeholders for other terms.

Recall also that the semantic substitutions ρ that are employed in the definition of the semantics of an NTSS must respect the sort of variables. This means that the rules above cannot be applied to prove term-for-atom substitution transitions. For example, given a complex (non-atom) term t, the transition $a \xrightarrow{a \overset{A}{\mapsto} t} t$ is not provable by the rule ($a1_{\text{As}}$).

As in the case for term-for-atom substitution, atom-for-atom substitution transitions are deterministic.

A syntactic atom-for-atom substitution over nominal terms, together with its corresponding correctness theorem, can be provided. It turns out to be just a straightforward adaptation of Theorem 5.4.2 and it is therefore omitted here[4].

5.4.2 α-conversion Transitions

The notion of α-conversion is a natural equivalence guaranteeing that the exact name chosen in binders is not important and can be indeed replaced by any other name (while avoiding capture). Unfortunately, not all names can be picked when performing this change. To exemplify this fact, we can consider again the λ-calculus. The term $\lambda a.(a\ b)$ is α-equivalent to $\lambda c.(c\ b)$ (provided that $a \neq b$ and $b \neq c$). The atom c is indeed a suitable atom for 'α-conversion'. However, the atom b is not suitable, because one does not want to α-convert the term $\lambda a.(a\ b)$ into $\lambda b.(b\ b)$ as this leads to the capture of a free atom. Again, thanks to freshness assertions, we can accommodate α-conversion in our framework as an ordinary transition relation. Given a nominal signature, the following deduction rules define $\rightarrow_{\alpha}$. For all atoms a and b and function symbols f, we have the following

[4] The corresponding correctness theorem is however stated explicitly in Section 5.14, as it is employed in the proof of Theorem 5.4.4 to follow. However, the proof for this theorem is omitted for it is a slight variant on the proof of Theorem 5.4.2.

rules.

$$x \to_\alpha x \ (\mathrm{id}_\alpha) \qquad \frac{x \xrightarrow{a \overset{A}{\mapsto} b} y \qquad b\#x}{[a]x \to_\alpha [b]y} \ (\mathrm{abs1}_\alpha) \qquad \frac{x \to_\alpha y}{[a]x \to_\alpha [a]y} \ (\mathrm{abs2}_\alpha)$$

$$\frac{\{x_i \to_\alpha x_i' \mid 0 < i \leq n\}}{f(x_1, x_2, \ldots, x_n) \to_\alpha f(x_1', x_2', \ldots, x_n')} \ (\mathrm{f}_\alpha) \qquad \frac{x \to_\alpha y \quad y \to_\alpha z}{x \to_\alpha z} \ (\alpha \cdot \mathrm{upTo}\alpha)$$

The reader will notice that α-conversion transitions rely on those for the atom-for-atom substitution (rule $(abs1_\alpha)$). Throughout the chapter, whenever we say that the rules above for α-conversion transitions are present in an NTSS, it implies that also the rules for atom-for-atom substitutions are present.

The reader is perhaps familiar with the syntactic version of α-conversion, defined below.

Definition 5.4.3 (α-conversion over nominal terms) *Let T be an NTSS. The relation $=_\alpha$ is the least congruence on nominal terms over the signature of T, such that, for all closed term M and an atom b, if b is fresh in M then $[a]M =_\alpha [b]M[b/a]$.*

The set of rules for α-conversion transitions generated above actually behaves according to the syntactic α-conversion, as stated in the following theorem.

Theorem 5.4.4 (Correctness of α-conversion transitions) *Let T be an NTSS. For all closed terms M and N over the signature of T, it holds that $M \to_\alpha N$ if and only if $M =_\alpha N$.*

The reader can find the proof of Theorem 5.4.4 in Section 5.14.

Calculi with binders usually consider a term as a representative of the equivalence class of all the terms that are α-convertible to it. In Nominal SOS, it is possible to achieve this by augmenting the NTSS with a deduction rule, given below.

Definition 5.4.5 (Transitions up to α-equivalence) *Let T be an NTSS and l be a label of the signature of T. The transition relation $\xrightarrow{l}$ is* up to α-equivalence *whenever the deduction rules of T contain the rules for α-conversion transitions, as defined above, the rules for atom-for-atom substitution transitions, as defined in Section 5.4.1, and the deduction rule:*

$$\frac{x \to_\alpha y \quad y \xrightarrow{l} z}{x \xrightarrow{l} z} \ (l \cdot \mathrm{upTo}\alpha).$$

Depending on the peculiarities of the calculus at hand, the modeller might want to consider defining some of the transition relations to be up to α-equivalence.

5.5 Examples

In this section we try to convince the reader of the expressiveness, and perhaps naturalness, of Nominal SOS by formulating in our framework two classical calculi, namely the lazy λ-calculus, [2], and the early π-calculus, [132, 98].

5.5.1 The lazy λ-Calculus

For ease of reference, we repeat here the signature given in Section 5.3 for λ-terms. The signature Σ^{λ} of our lazy λ-calculus is constructed using a base sort L for λ-terms and an atom sort A and the following function symbols.

1. $\lambda(\_) : [A]L \to L$: A unary function symbol for embedding abstractions inside terms;
2. $\_\ \_ : (L \times L) \to L$: A binary function symbol for application.

The semantics includes a reduction transition $\to$, here displayed with no label to remain in line with the standard notation from [2], and the rules for term-for-atom substitution transitions as generated in Section 5.4.1.

The set of rules of the signature Σ^{λ} contains the following two derivation rules, which define the operational semantics of our version of the lazy λ-calculus, for all atoms a.

$$\frac{}{\lambda([a]x) \to \lambda([a]x)}\ (\text{abs}) \qquad \frac{x_0 \to \lambda([a]y_0) \quad y_0 \overset{a \overset{T}{\mapsto} x_1}{\longrightarrow} y_1 \quad y_1 \to y_2}{(x_0\ x_1) \to y_2}\ (\text{app})$$

Moreover, the transition $\to$ and the term-for-atom substitution transitions are up to α-equivalence. The reader may want to notice that by Definition 5.4.5 this means that the set of rules of Σ^{λ} contains also the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. The rules for atom-for-atom substitution transitions are also set to be up to α-equivalence.

Some definition will be useful in the next sections. In particular, we say that a term $M \in \mathbb{T}(\Sigma^\lambda)$ has a *normal form* whenever it holds that $M \to M'$ for some M'. In that case, M' is a normal form for M. For instance, every abstraction has a normal form while the term $(\lambda([a](a\ a))\ \ \lambda([a](a\ a)))$ has no normal form.

We denote by Λ be the set of λ-terms of [2, 33]. The encoding $[\![\cdot]\!]^\lambda$ is a map from Λ into terms of our nominal λ-calculus and it is defined as follows.

$$\begin{array}{rcl} [\![a]\!]^\lambda & = & a \\ [\![\lambda a.M]\!]^\lambda & = & \lambda([a][\![M]\!]^\lambda) \\ [\![(M\ N)]\!]^\lambda & = & ([\![M]\!]^\lambda\ [\![N]\!]^\lambda) \end{array}$$

We also present the rules for the semantics of the original lazy λ-calculus, as recalled from [2][5].

$$\frac{}{\lambda a.x \to \lambda a.x}\ (\text{absO}) \qquad \frac{x_0 \to \lambda a.y_0 \quad y_0[x_1/a] \to y_1}{(x_0\ x_1) \to y_1}\ (\text{appO})$$

The reader should not confuse the substitution operation over λ-terms employed in the rule (appO) with the syntactic substitution over nominal terms defined in Section 5.4.1. Also, λ-terms are considered up to α-equivalence. The substitution and α-equivalence of λ-calculus are standard and not provided in the main body of the chapter. However, for completeness, they are repeated in Section 5.11 together with some other useful definitions.

We now have all the ingredients to state a correspondence between the two calculi. The following theorem establishes the operational correctness of our formulation of the lazy λ-calculus with respect to its original formulation.

Theorem 5.5.1 (Operational Correspondence: lazy λ-calculus) *For all $M, N \in \Lambda$, $M \to N \Leftrightarrow [\![M]\!]^\lambda \to [\![N]\!]^\lambda$.*

The proof of Theorem 5.5.1 is contained in Section 5.15.

5.5.2 The early π-calculus

The signature Σ^π of our π-calculus is modelled by a base sort P and atom sort C (for processes and channels, respectively) and the following function symbols.

1. $\mathbf{0} :\to P$ for inaction (deadlock),

[5] In [2], Abramsky uses the symbol $\Downarrow$ in lieu of $\to$.

2. $\tau.\_ : P \to P$ for τ-prefix,
3. $out(\_,\_,\_) : (C \times C \times P) \to P$ for output prefix,
4. $in(\_,\_) : (C \times [C]P) \to P$ for input prefix,
5. $\nu(\_) : [C]P \to P$ for restriction,
6. $\_ \mid \_ : (P \times P) \to P$ for parallel composition,
7. $\_ + \_ : (P \times P) \to P$ for nondeterministic choice,
8. $!\_ : P \to P$ for parallel replication.

The reader should notice that since our aim in this section is to provide an example of usage of Nominal SOS to the user, we prefer to stick with the exact definitions of the framework, and the syntax employed for input and output prefix differs slightly from the standard notation used in the π-calculus. In particular, our term $out(a, b, P)$ corresponds to the process $\overline{a}b.P$ of the π-calculus, and $in(a, [b].P)$ corresponds to $a(b).P$. The same choice is adopted for the labels.

Below, we specify the semantics of the early π-calculus in Nominal SOS. Since in our framework labels are open terms, we display an input transition label as $in(a, b)$, assuming a different operator in accepting two atoms as arguments. For presentational purposes, we use the same names to stipulate the meaning of the transitions. For the same reasons, we model an output transition label as $out(a, b)$ and a bound output transition label as $bout(a, [b]\mathbf{0})$, abbreviated as $bout(a, [b])$ throughout the text. The constant τ is also used in τ-transitions. The set of rules of the signature Σ^{π} contains the following rules, we use α to range over labels and a, b and c to range over atoms.

$$\frac{}{\tau.x \xrightarrow{\tau} x}\ (\tau) \qquad \frac{}{out(a,b,x) \overset{out(a,b)}{\rightarrow} x}\ (\text{out}) \qquad \frac{x \overset{b \overset{A}{\mapsto} c}{\longrightarrow} y}{in(a,[b]x) \overset{in(a,c)}{\rightarrow} y}\ (\text{in})$$

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1}\ (\text{sum1}) \qquad \alpha \notin \{bout(a,[b]) \mid a,b \in C\}\ \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \parallel x_2 \xrightarrow{\alpha} y_1 \parallel x_2}\ (\text{par1})$$

$$\frac{x_1 \overset{bout(a,[b])}{\rightarrow} y_1 \quad b\#x_2}{x_1 \parallel x_2 \overset{bout(a,[b])}{\rightarrow} y_1 \parallel x_2}\ (\text{parRes1}) \qquad \frac{x_1 \overset{out(a,b)}{\rightarrow} y_1 \quad x_2 \overset{in(a,b)}{\rightarrow} y_2}{x_1 \parallel x_2 \xrightarrow{\tau} y_1 \parallel y_2}\ (\text{com1})$$

$$\frac{x_1 \overset{bout(a,[b])}{\rightarrow} y_1 \quad x_2 \overset{in(a,b)}{\rightarrow} y_2 \quad b\#x_2}{x_1 \parallel x_2 \xrightarrow{\tau} \nu([b](y_1 \parallel y_2))}\ (\text{close1}) \qquad \frac{x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y \parallel !x}\ (\text{repl})$$

$$\frac{x \overset{out(z,a)}{\rightarrow} y \quad a \neq z}{\nu([a]x) \overset{bout(z,[a])}{\rightarrow} y}\ (\text{open}) \qquad c \notin ba(\alpha)\ \frac{x \xrightarrow{\alpha} y \quad c\#\alpha}{\nu([c]x) \xrightarrow{\alpha} \nu([c]y)}\ (\text{res})$$

For the sake of brevity, we omit the symmetric versions of rules (sum1), (par1), (parRes1), (com1) and (close1). These are referred to as (sum2), (par2), (parRes2), (com2) and (close2). Moreover, for each label l, $\overset{l}{\rightarrow}$ is up to α-equivalence. As in the previous case, by Definition 5.4.5 this means that the set of rules of Σ^{λ} contains the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. The latter are needed both in the context of α-conversion and in the rules which specifically define our formulation of π-calculus, see rule (in). We set the atom-for-atom transition relations to be up to α-equivalence, too.

The reader must notice that the complicated side-conditions of the ordinary formulation of π-calculus are here replaced by rather simpler freshness conditions, see rules (parRes1) and (close1). By way of example, we consider for instance the rule of the original π-calculus that describes the interleaving semantics of the parallel composition operator. The complete set of rules is, however, given later on in this section.

$$bn(\alpha) \cap fn(x_2) = \emptyset\ \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \parallel x_2 \xrightarrow{\alpha} y_1 \parallel x_2}\ (\text{parO1})$$

where $bn(\alpha)$ denotes the set of bound names that occur in the label α and $fn(P)$ denotes the set of names that have a free occurrence in P, see [132]. (They are however repeated in Section 5.12.)

The side-condition for this rule is "external" to the semantic specification. In implementations, checking whether this side-condition holds is handled only by a run-time check, after a substitution is applied in order to use the rule (parO1) with concrete terms. On the theoretical side another approach is to populate the SOS semantics with copies of the same rule for each combination of label, substituted to α, and closed term, substituted for x_2, that satisfies the side-condition. This approach does not simplify the theory. As a result, proofs in the context of the π-calculus are usually non-uniform in their development, having the necessity to switch, from time to time, from technicalities within the semantics of the calculus to technicalities concerning the external level of the side-conditions.

In our formulation of π-calculus, the behaviour of the rule (parO1) is modelled by considering two cases. In the first case we define the interleaving semantics of the parallel composition for all the labels that do not need the considered special checking. These are all the labels with the exception of the labels regarding bound outputs. Rule (par1) is devoted to this aim. In the second case, we manage the labels that require the extra checking, namely the bound outputs. Thanks to the freshness premises, we are able to model this checking within the framework, see rule (parRes1).

The convenient use of freshness premises in the rules for π-calculus shows, in some sense, that the novelty of having freshness tests in rules is not only useful in modelling in a direct way meta-level operations such as substitutions and α-conversion, see Section 5.4, but it is also useful in the modelling of specific features of languages.

We denote by Π be the set of π-terms of [132]. The encoding $[\![\cdot]\!]^\pi$ is a map from Π into terms of our nominal π-calculus and it is defined as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]^\pi &= \mathbf{0} \\
[\![\tau.P]\!]^\pi &= \tau.[\![P]\!]^\pi \\
[\![\overline{a}b.P]\!]^\pi &= out(a, b, [\![P]\!]^\pi) \\
[\![a(b).P]\!]^\pi &= in(a, [b][\![P]\!]^\pi) \\
[\![va.P]\!]^\pi &= \nu([a][\![P]\!]^\pi) \\
[\![P + Q]\!]^\pi &= [\![P]\!]^\pi + [\![Q]\!]^\pi \\
[\![P \parallel Q]\!]^\pi &= [\![P]\!]^\pi \parallel [\![Q]\!]^\pi \\
[\![!P]\!]^\pi &= ![\![P]\!]^\pi
\end{aligned}$$

Since we use a different notation for actions, the encoding is extended to labels as follows.

$$\begin{aligned} [\![\tau]\!]^{\pi} &= \tau \\ [\![ab]\!]^{\pi} &= in(a,b) \\ [\![\overline{a}b]\!]^{\pi} &= out(a,b) \\ [\![\overline{a}(b)]\!]^{\pi} &= bout(a,[b]) \end{aligned}$$

For ease of reference, we repeat below the rules of the semantics of the original early π-calculus of [132][6]. In the rules that follow, $names(\alpha)$ denotes the set of names that occur in the label α, $bn(\alpha)$ denotes the set of bound names that occur in the label α and $fn(P)$ denotes the set of names that have a free occurrence in P, see [132]. These definitions are repeated in Section 5.12 together with some other useful definitions from the standard theory of π-calculus.

$$\frac{}{\tau.x \xrightarrow{\tau} x}\ (\tau\text{O}) \qquad \frac{}{\overline{a}b.x \xrightarrow{\overline{a}b} x}\ (\text{outO}) \qquad \frac{}{a(b).x \xrightarrow{ac} x[c/b]}\ (\text{inO})$$

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1}\ (\text{sumO1}) \qquad bn(\alpha) \cap fn(x_2) = \emptyset\ \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \parallel x_2 \xrightarrow{\alpha} y_1 \parallel x_2}\ (\text{parO1})$$

$$\frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \parallel x_2 \xrightarrow{\tau} y_1 \parallel y_2}\ (\text{comO1})$$

$$b \notin fn(x_2)\ \frac{x_1 \xrightarrow{\overline{a}(b)} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \parallel x_2 \xrightarrow{\tau} \nu b.(y_1 \parallel y_2)}\ (\text{closeO1}) \qquad \frac{x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y \parallel !x}\ (\text{replO})$$

$$a \neq z\ \frac{x \xrightarrow{\overline{z}a} y}{\nu a.x \xrightarrow{\overline{z}(a)} y}\ (\text{openO}) \qquad c \notin names(\alpha)\ \frac{x \xrightarrow{\alpha} y}{\nu c.x \xrightarrow{\alpha} \nu c.x}\ (\text{resO})$$

For the sake of brevity, we omit the symmetric versions of rules (sumO1), (parO1), (parResO1), (comO1) and (closeO1). In what follows, these are referred to as (sumO2), (parO2), (parResO2), (comO2) and (closeO2).

The reader should not confuse the opertation of substitution over π-terms employed in the rule (inO) with the syntactic substitution over nominal terms defined in Section 5.4.1. Also, π-terms are considered up to α-equivalence. The defini-

[6] [132] presents more rules for the replication operator for technical reasons we are not concerned with here, see pages 42 and 43 in that reference for more information.

tions of substitution and α-equivalence of the π-calculus are standard and not provided in the main body of the chapter. (They are however repeated in Section 5.12.)

We now have all the ingredients to state a correspondence between the two calculi. The following theorem establishes that our formulation of the early π-calculus is operationally correct with respect to its original formulation.

Theorem 5.5.2 (Operational Correspondence: early π-calculus) *For all $P, Q \in \Pi$, $P \xrightarrow{\alpha} Q \Leftrightarrow [\![P]\!]^{\pi} \xrightarrow{[\![\alpha]\!]^{\pi}} [\![Q]\!]^{\pi}$, where α ranges over the labels of the form τ, ab, $\overline{a}b$ and $\overline{a}(b)$ from the original early pi-calculus.*

The proof of Theorem 5.5.2 is contained in Section 5.16.

5.5.3 A remark on the Barendregt Convention

Typically, in calculi with binders, terms are assumed to make use of some discipline on the choice of the names for the variables used in programs. This discipline is the so called *Barendregt Convention* and roughly states that, within a term, the names of bound variables must be chosen to be all different and also different from every free variable. For instance, in the context of the λ-calculus, the term $(\lambda x.(x\ y)\ x)$ is not considered part of the λ-calculus by the Barendregt Convention. This term is considered only in one of its α-equivalent forms that does respect the convention, for instance the term $(\lambda z.(z\ y)\ x)$.

This discipline on the names for variables is very useful in the development of the theory of calculi with binders. In particular it prevents the user from dealing with technicalities due to name clashes and allows one to focus directly on the computational aspects of a calculus. Of course, when implementing a calculus, the Barendregt Convention must, however, be addressed with a pre-processing step.

Our formulations of the lazy λ-calculus and of the early π-calculus do not make use of this convention; in fact, we do not make use of *any* convention regarding the name of atoms used in the terms.

Considering, for instance, our formulation of the lazy λ-calculus, the user can write the term $(\lambda([a].(\lambda([b].a\ b)))\ b)$. This term obviously does not adhere to the Barendregt Convention, as the atom b is used both as bound and free. The reader may notice however that this is not a problem in our formula-

tion of the λ-calculus. Indeed, the term is automatically converted to an α-equivalent *good* one at the moment of performing a computational step. A transition $(\lambda([a].(\lambda([b].a\ b)))\ b) \rightarrow \lambda([c].b\ c)$ is indeed provable by rule (app), for all atoms c that are fresh in the subterm $(a\ b)$ of $\lambda([b].a\ b))$. In more detail, what happens in this situation is that the premise $y_0 \xrightarrow{a \overset{T}{\mapsto} x_1} y_1$ of the rule (app) cannot be satisfied in general when y_0 is instanciated for the term $(\lambda([b].a\ b)))\ b)$. Indeed, without α-conversion, the term $(\lambda([b].a\ b)))\ b)$ does not perform a transition $\xrightarrow{a \overset{T}{\mapsto} b}$ since the bound atom b is not fresh in the argument term b, and the premise $a\#z$ of $(abs1_{\mathrm{Ts}})$, instanciated in that context as $b\#b$, would not be satisfied. Fortunately, we set the term-for-atom substitution transitions to be up to α-equivalence, so one instance of the rule

$$\frac{x \rightarrow_{\alpha} y \quad y \xrightarrow{a \overset{T}{\mapsto} b} z}{x \xrightarrow{a \overset{T}{\mapsto} b} z}\ (a \overset{T}{\mapsto} b \cdot \mathrm{upTo}\alpha).$$

is actually employed. The term $(\lambda([b].a\ b)))\ b)$ is thus α-converted to another term, which actually allows the premise $a\#z$ of $(abs1_{\mathrm{Ts}})$ to be satisfied.

The nominal machinery employed here plays a crucial role in this scenario, allowing for an implicit search for a suitable fresh atom, which is clearly guaranteed to exist.

In some sense, Nominal SOS naturally implements a sort of lazy Barendregt Convention, i.e. the change of atom names into suitable ones is performed during the computational steps, on demand, and only when facing the name clashes. All of this is possible thanks to the adoption of the nominal approach.

In implementations, these nominal calculi formalized using Nominal SOS do not require any pre-processing step in order to change the name of atoms in programs. This is particularly desirable in distributed contexts, where different pieces of code may come from different locations, written by different programmers who have no idea about the names for bound atoms used by others, and still their programs must be combined together.

For the sake of clarity, we point out that, by not assuming the Barengret Convention, the user can also write terms such as $(\lambda([a].(\lambda([a].a)\ a)\ b)$. In this case the computational step behaves as expected. In particular we can prove the transi-

tion $(\lambda([a].(\lambda([a].a)\ a)\ b) \rightarrow (\lambda([a].a)\ b)$, where the term $\lambda([a].a)$ remains unchanged because of the transition $\lambda([a].a) \xrightarrow{a \overset{T}{\mapsto} b} \lambda([a].a)$ proved using rule $(abs2_{\text{Ts}})$.

5.6 Nominal bisimilarity

Very often, in the theory of calculi with binders, the ordinary bisimilarity is not a satisfactory equivalence. This is typical in calculi that allow terms to perform transitions whose labels mention fresh or bound variables. As a prominent example, we show the following example taken from [132] in the context of the π-calculus. We first recall the definition of bisimilarity over π-calculus processes, for which we overload the symbol $\leftrightarrows$.

Definition 5.6.1 (Bisimilarity) Bisimilarity $\leftrightarrows$ *is the largest symmetric binary relation* $\sim$ *on* Π *such that whenever* $P \sim Q$*, for all labels* l *it holds that if* $P \xrightarrow{l} P'$ *then there exists* Q'*, such that* $Q \xrightarrow{l} Q'$ *and* $P' \sim Q'$*.*

With respect to bisimilarity, the processes

$$\begin{aligned} P &= \nu z.\overline{x}z. \\ Q &= \nu z.(\ \overline{x}z.\ \|\ \nu w.\overline{w}y.\ ) \end{aligned}$$

are distinguished[7]. Indeed, since processes in the π-calculus are considered up to α-equivalence, we have that $P \xrightarrow{\overline{x}(y)}$. On the other hand Q can not turn its binder $x(z)$ into $x(y)$ by α-conversion, because y is one of its free variables. Therefore $Q \overset{\overline{x}(y)}{\not\rightarrow}$. Of course, P and Q should not be distinguished, and what actually happens in the theory of the π-calculus is that the transitions of the form $\xrightarrow{\overline{x}(z)}$ from P and Q are matched only for those variables z that are free in *both* P and Q. The bisimulation game is thus modified, and not all of the transitions from P and Q are considered in the matching process[8].

In what follows, we define this modified type of bisimilarity, here called *nominal bisimilarity*. In doing so, the reader should bear in mind that Nominal SOS models labels as open terms. In our framework, we therefore require that the bisimilarity would match labels containing abstractions only when the bound atoms are fresh in both of the considered terms. For instance, the bisimilarity in our formulation

[7] The process $\nu w.\overline{w}y.$ performs an output on a channel which is not known by the external environment, therefore this process is bisimilar to $\mathbf{0}$.

[8] This point is explained carefully on pages 64-65 of [132].

of the π-calculus will try to match bound output transitions of P and Q only for labels of the form $bout(a, [b]\mathbf{0})$ where the atom b is fresh in both P and Q.

Definition 5.6.2 (Nominal bisimilarity) *Let T be an NTSS.* Nominal bisimilarity $\leftrightarrows_T$ *is the largest symmetric binary relation $\sim$ over closed terms of T such that whenever $P \sim Q$, for all labels l such that for all $a \in ba(l)$, $a\#P$, $a\#Q$, it holds that if $P \xrightarrow{l} P'$ then there exists Q', such that $Q \xrightarrow{l} Q'$ and $P' \sim Q'$.*

In what follows we will always omit the subscript in $\leftrightarrows_T$, and write $\leftrightarrows$ for nominal bisimilarity, as T will be always clear from the context.

We prove that what the nominal bisimilarity does in the ordinary π-calculus is exactly what bisimilarity does in our formulation of the π-calculus when ignoring the matching of substitution transitions in the bisimulation game. We denote this equivalence with $\underline{\leftrightarrow}^-$. In what follows the encoding $[\![\cdot]\!]^\pi$ is the mapping defined in Section 5.5.2.

Theorem 5.6.3 (Nominal bisimilarity is bisimilarity, when ignoring substitutions) *For all $P, Q \in \Pi$, $P \leftrightarrows Q$ if, and only if, $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$.*

The proof of Theorem 5.6.3 can be found in Section 5.21.

The reader may wonder what is the equivalence over π-calculus terms that corresponds to nominal bisimilarity, including substitution transitions. The next theorem provides us with an answer: nominal bisimilarity in our formulation of the early π-calculus coincides with Sangiorgi's open bisimilarity, see [132, Section 4.2] and [131]. The open bisimilarity involves substitutions, which are defined as follows, as recalled from [132] (Definition 1.1.3 on page 14).

Definition 5.6.4 (Substitutions involved in the open bisimilarity) *A* substitution *σ is a function on names that is the identity except on a finite set.*

In this section we use the symbol σ for substitutions, as it is standard in the literature on the π-calculus. The reader should not however confuse substitutions with the sorts of Definition 5.2.1.

We now recall the definition of open bisimilarity from [132, 131].

Definition 5.6.5 (Open Bisimilarity) Open bisimilarity $\leftrightarrows_o$ *is the largest symmetric relation $\sim$ on Π such that whenever $P \sim Q$, and σ is a substitution (of Definition 5.6.4), if $P\sigma \xrightarrow{\alpha} P'$, then there exists Q', such that $Q\sigma \xrightarrow{\alpha} Q'$ and $P' \sim Q'$.*

The reader should notice that this definition is the very basic formulation of open bisimilarity, which does not involve distinctions, see [132] and [131]. In Definition

5.6.5, it is important to note that the ranging over all the substitutions is performed at each step of the bisimulation game.

Note furthermore that the substitutions involved in the open bisimilarity are capture avoiding. This is a necessary constraint, for open bisimilarity would not be a satisfactory equivalence otherwise. The reader can indeed consider the two terms

$$\begin{aligned} P &= \nu a.\overline{b}b. \\ Q &= \nu c.\overline{b}b. \end{aligned}$$

and the substitution σ that maps the name b to a and is the identity over all the other names. If we allowed the application of substitutions to capture free names, by applying σ to the two terms above we have that $P\sigma = \nu a.\overline{a}a.$, which is nominal bisimilar to $\mathbf{0}$, and $Q\sigma = \nu c.\overline{a}a.$, which can perform an output on the channel a. In this scenario, the terms P and Q would be thus distinguished, even though they are α-equivalent.

Theorem 5.6.6 (Open bisimilarity and Bisimilarity coincide) *For all $P, Q \in \Pi$, $P \mathrel{\reflectbox{$\leftrightarrows$}} Q$ if, and only if, $[\![P]\!]^{\pi} \leftrightarrows [\![Q]\!]^{\pi}$.*

The proof of Theorem 5.6.6 can be found in Section 5.22.

In passing, we briefly discuss the version of open bisimilarity that involves *distinctions*. Indeed, the open bisimilarity as in Definition 5.6.5 is not a satisfactory equivalence for the π-calculus, as it does not take into account the fact that two atoms that are bound by a restriction ν can never be identified. This point is made clear on page 167 of [132] by means of an example which we shall now show. Let us consider the two processes P and Q defined as follows[9]:

$$\begin{aligned} P &= \nu z.\nu w.\overline{x}z.\overline{x}w.\ (\overline{z} \parallel w) \\ Q &= \nu z.\nu w.\overline{x}z.\overline{x}w.\ (\overline{z}.w + w.\overline{z}). \end{aligned}$$

We have that P and Q are not open bisimilar. To see this, it suffices to note that the subterm $(\overline{z} \parallel w)$ of P and the subterm $(\overline{z}.w + w.\overline{z})$ of Q are not open bisimilar. Indeed, if we fix a substitution σ that maps w to z and is the identity over all the other channel names, we have that $(\overline{z} \parallel w)\sigma \xrightarrow{\tau}$ and $(\overline{z}.w + w.\overline{z})\sigma \not\xrightarrow{\tau}$.

However, both the channel names z and w occur underneath a ν restriction and they are thus forced to be distinct. Therefore, no substitution can make the two channels suddenly able to communicate with each other.

[9] In order to present our point more clearly, the two processes $(\overline{z} \parallel w)$ and $(\overline{z}.w + w.\overline{z})$ employ a CCS-style syntax.

The processes P and Q should indeed be considered equivalent. Sangiorgi equips the open bisimilarity with distinctions in [131], see also [132], that keep track of the names that cannot be identified during the bisimulation game. The open bisimilarity with distinctions is a satisfactory equivalence for the π-calculus, as argued in [132].

However, the scenario discussed above is too dependent on the intrinsic meaning of binders to be captured by a general theory like the one we put forward in this chapter. For instance, in the π-calculus there are two binders $va.P$ and $a(b).P$, but only the former populates distinctions. The latter binder has a different meaning. Since this work addresses only a basic and uniform account of binders, providing an adaptation of the nominal bisimilarity of Definition 5.6.2 in order to tackle distinctions, although possible, is not considered in this first development and is part of our future work.

5.7 Applicative Bisimilarity

In the context of the lazy λ-calculus, one of the most interesting notions of bisimilarity is the *applicative bisimilarity* due to Samson Abramsky, [2]. Below we recall the definition of this equivalence. We also recall that a λ-term is closed if it contains no free variables, and that we denote the set of closed λ-terms by Λ^0.

Definition 5.7.1 (Applicative Bisimilarity in the λ-calculus) *The* applicative bisimilarity *is the largest symmetric relation $\simeq$ on Λ^0 such that whenever $M \simeq N$, if $M \to \lambda a.M'$ for some variable a and $M' \in \Lambda$, then there exist some variable b and $N' \in \Lambda$ such that*

- $N \to \lambda b.N'$, *and*
- $M'[P/a] \simeq N'[P/b]$, *for all $P \in \Lambda^0$.*

It is important to remark that the applicative bisimilarity of the λ-calculus is defined over closed λ-terms. Indeed, this equivalence is strongly unsatisfactory over terms that contain free variables. For instance, for any variables a, b and c, it holds that $a \simeq b$ and $\lambda a.b \simeq \lambda a.c$.

The notion of applicative bisimilarity can be accommodated within the context of Nominal SOS in the obvious way.

Definition 5.7.2 (Applicative Bisimilarity over $\mathbb{C}(\Sigma^\lambda)$) *The* applicative bisimilarity *is the largest symmetric relation $\simeq$ on $\mathbb{C}(\Sigma^\lambda)^0$ such that whenever $M \simeq N$, if*

$M \to \lambda([a].M')$ for some atom a and $M' \in \mathbb{C}(\Sigma^\lambda)$, then there exist some atom b and $N' \in \mathbb{C}(\Sigma^\lambda)$ such that

- *$N \to \lambda([b].N')$, and*
- *$M'[P/a] \simeq N'[P/b]$, for all $P \in \mathbb{C}(\Sigma^\lambda)^0$.*

The following theorem states that the applicative bisimilarity of our formulation of the lazy λ-calculus and that of its original formulation coincide.

Theorem 5.7.3 (Applicative Bisimilarity in Λ and in Σ^λ coincide) *For all $M, N \in \Lambda^0$, $M \simeq N$ if, and only if, $[\![M]\!]^\lambda \simeq [\![N]\!]^\lambda$.*

The proof of Theorem 5.7.3 follows easily from Theorem 5.5.1, which states the operational correctness of our lazy λ-calculus with respect to the original calculus.

It is now natural to wonder what are the relationships between the applicative and the nominal bisimilarity in our formulation of the lazy λ-calculus. Unfortunately, nominal bisimilarity is a very unsatisfactory equivalence over $\mathbb{C}(\Sigma^\lambda)^0$. In particular, all the terms in $\mathbb{C}(\Sigma^\lambda)^0$ that have a normal form are equated.

Theorem 5.7.4 (Nominal Bisimilarity equates all 'terminating' terms) *For all $M, N \in \mathbb{C}(\Sigma)^0$, if M and N have a normal form then $M \leftrightarrows N$.*

The proof of Theorem 5.7.4 can be found in Section 5.24.

Notice moreover, that both nominal and applicative bisimilarity equate all the terms that do not have a normal form. The implications of Definitions 5.6.2 and 5.7.2 are indeed vacuously true. From this fact and from Theorem 5.7.4, the following theorem easily follows.

Theorem 5.7.5 (Applicative is included in the nominal bisimilarity) *It holds that $\simeq \subset \leftrightarrows$.*

However, the applicative bisimilarity is not included in the nominal bisimilarity. This is stated in the following theorem.

Theorem 5.7.6 (Nominal is *not* included in the applicative bisimilarity) *It holds that $\leftrightarrows \not\subset \simeq$.*

Theorem 5.7.6 follows easily from the fact that applicative bisimilarity distinguishes some binding-closed terms that have a normal form. Any two such terms are counter-example witnesses for Theorem 5.7.6. By way of example, let us consider the two terms $M = \lambda([a].a)$ and $N = \lambda([a].(a\ a))$. Notice first that M and

N are binding-closed and they have a normal form. By Theorem 5.7.4 we have therefore that $M \leftrightarrows N$. However, it holds that $M \not\simeq N$. To see this, the reader should notice that after the 'parameter passing' of the term N, the two terms are distinguished. Namely, at the second step of the applicative bisimilarity we are required to check, among other checks, whether the two terms $a[N/a] = \lambda([a].(a\ a))$ and $(a\ a)[N/a] = (\lambda([a].(a\ a))\ \ \lambda([a].(a\ a)))$ are applicative bisimilar. This is not the case, as the former has a normal form while the latter does not have a normal form. We have that $\lambda([a].(a\ a)) \not\simeq (\lambda([a].(a\ a))\ \ \lambda([a].(a\ a)))$ and consequently $M \not\simeq N$.

From the fact that the applicative bisimilarity of our lazy λ-calculus coincides with that of its original formulation (Theorem 5.7.3), the following theorem follows as a straightforward consequence of Theorems 5.7.5 and 5.7.6.

Theorem 5.7.7 (Nominal and Applicative Bisimilarity relation through the encoding) *For all $M, N \in \Lambda^0$,*

- *$M \simeq N$ implies $[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$, and*
- *$[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$ does* not *imply $M \simeq N$.*

5.8 Related Work

Nominal SOS is not the only approach studied so far in the literature that aims at a uniform treatment of binders and names in the operational semantics of programming and specification languages. We are aware of a number of existing approaches that accommodate variables and binders inside the SOS framework, namely those by Fokkink and Verhoef in [59], by Middelburg in [90, 91], by Miller and Tiu in [93] and by Fiore and Staton in [56] (originally, by Fiore and Turi [57]). Moreover, Gacek, Miller and Nadathur, in [64], Lakin and Pitts in [85] (MLSOS) and also in [86, 87, 84] (αML), and Sewell at al. in [135] have also proposed formalization of binders within SOS-like frameworks. In the following sections we discuss in some detail the approaches that are more related to our work.

5.8.1 MLSOS and αML

In [85], Lakin and Pitts provide the meta-language MLSOS in which the various transition relations of SOS can be specified in a functional style language. In this section we do not repeat the definitions of the system from [85]. However,

a reader who is familiar with functional programming will have no difficulty in understanding the code fragments we present.

In [85], the authors provide a formulation of the λ-calculus that employs the parallel reduction strategy, i.e. the reduction is performed in any context. Below we show a formulation of the simpler call-by-name λ-calculus, see [33]. Intuitively, in the call-by-name λ-calculus the reduction is inductively shared only to the first argument of an application. Namely, if $M \to M'$, then $(M\ N) \to (M'\ N)$. The π-calculus is not formalized in [85] and therefore we do not provide code for it. The example for the λ-calculus suffices to show to the reader how MLSOS provides a language for the specification of the SOS semantics of calculi with binders.

```
1: nametype var;;
2: datatype lam =
3:       Var of var
4:       Lam of <<var>> lam
5:       App of lam * lam;;
6:
7: let rec sub x t t' = narrow t' as
8:       Var y:  ((x=:=y);t or ((x=/=y : var);t')
9:       Lam <<a>>t'':   Lam <<a>>sub x t t'':
10:      App (t_1 t_2):  App ((sub x t t_1) (sub x t t_2));;
11:
12: let rec beta (t_1 t_2) = narrow (t_1 t_2) as
13:      (t t):           yes
14:      (App (t_1 t_2), t_3):   beta(t_1 t_2)
15:      (App ((Lam <<a>>t_1), t_2), t'):
16:              some t_1', t_2': lam in
17:              sub(a, t_2', t_1') =:= t';;
```

Lines 1-5 define the grammar for λ-terms, lines 7-10 define the substitution procedure and lines 12-17 defines the reduction relation. The reader can see how the way of defining the terms of a calculus recalls the way of defining inductive types in a functional language. Also, the definition of the various transition relations are encoded as some sort of function definitions of functional languages.

It is important however to notice that, in MLSOS, the only operations allowed on atoms are equality tests and the generation of fresh atoms. In MLSOS, in a λ-term

$$\lambda\langle\langle a\rangle\rangle(Var\ a)$$

the variable a actually ranges over the infinite set of atoms, and will be instanciated to a fresh atom on demand. The user cannot name atoms directly.

This approach stems directly from Nominal Logic [117]. In particular, the nominal $\mathcal{N}$-quantifier is employed in this case, see also [61] where it was introduced. This solution to the modeling of languages was adopted already in FreshML, [137, 136]. FreshML adds to ML the capability to generate fresh new names with the keyword **fresh**. For instance, in the term $let\ x\ =\ \textbf{fresh}\ in\ Lam\langle x\rangle(Var\ x)$, the occurrences of the keywork **fresh** are evaluated by generating a name never seen before and placed in the body of the abstraction. The management of this, and the actual atom names employed in the execution of a program, are hidden to the user.

A different choice is instead made in the meta-language αML, [86, 87], which, in a sense, is what MLSOS evolved into. The syntax of αML shares many similarities with the one of MLSOS, in particular, it stems from its functional language style syntax. However, some peculiarities of αML also come from the world of the constraint logic programming, [81]. The management of variables of αML follows a different philosophy. As argued in Lakin's Thesis [84], where an excellent and complete overview of the language αML can be found, in the term

$$Lam\langle\langle x\rangle\rangle(Lam\langle\langle y\rangle\rangle(Var\ x))$$

the variables x and y are not assumed to be distinct just for having a distinct name. This term can denote both the λ-terms $\lambda a.\lambda b.a$ and $\lambda a.\lambda a.a$, as y is not required to denote a name that is different to the one denoted by x.

One can however state explicitly some conditions on name. In particular, the user can disseminate the program with constraints of name equality and freshness. If we wish the variables x and y to denote different names in the term above, we can rewrite it in αML as

$$\exists x : var.\exists y : var.(x\#y\ \&\ Lam\langle\langle x\rangle\rangle(Lam\langle\langle y\rangle\rangle(Var\ x))).$$

The management of constraints is delegated to constraint solving facilities of constraint logic programming. The interested reader is invited to refer to Lakin's Thesis [84].

The reader must notice that the handling of variables in all of the mentioned approaches (MLSOS, FreshML and αML) is different from how Nominal SOS handles name. In particular, in those approaches the user does not have access

to concrete atoms/variables directly. This goes along with what in the nominal context is called *the equivariance property*, see [61] and especially [117]. This property roughly states that the meaning of programs should not depend upon the concrete atom name which is chosen in the implementation of abstractions. On the other hand, within Nominal SOS one can write programs whose behaviour is strongly associated to a particular bound name. For instance, let us consider augmenting the rules of our formulation of the lazy λ-calculus with the following rule, where N is a given closed term and a is a fixed atom. (The following rule is therefore not replicated for all atoms and closed terms.)

$$\frac{x \xrightarrow{a \mapsto N} x'}{\lambda([a]x) \rightarrow x'}$$

In this case, the calculus acts non-deterministically for λ-abstraction terms that use the atom a as bound. Namely, given a closed term M, the term $\lambda([a]M)$ can perform two transitions. We can apply the rule (abs) of Section 5.5.1 in order to prove the standard transition $\lambda([a]M) \rightarrow \lambda([a]M)$. On the other hand, we can apply the rule above. Applying this rule, the abstraction $\lambda([a]M)$ pretends that the parameter passing of a term N took place. Since the rule above is defined only for the atom a, the rule above cannot be applied for a term $\lambda([b]M)$, with a and b distinct atoms.

Defining the behaviour of programs in a way that it depends on specific bound names should be considered bad language design. In Nominal SOS, however, a language specifier is free to explore also this possibility.

Another difference between Nominal SOS and the approaches of MLSOS and αML is that the latter ones provide for a primitive built-in support for managing the abstract syntax trees of terms which are identified up to α-equivalence. The motivation for such a choice is that this support for α-equivalence is required in any calculus that contains binders. If α-equivalence is not built-in, this support needs to be coded up by hand. In our context, the code for this management does not represent much of a problem; indeed it can be automatically generated, as shown in Section 5.4. Secondly, as our design choice, we prefer α-equivalence to be a transition in the semantics just like any other, so that it can be the subject of meta-theorems based on the shape of rules that may be developed in the future.

5.8.2 $FO\lambda^{\Delta\nabla}$

In [93], Miller and Tiu make use of a logic that is already equipped with λ-terms; this approach is called *the λ-tree approach* to encoding syntax. The particular logic they employ is called $FO\lambda^{\Delta\nabla}$ (fold-nabla). The main peculiarity of this logic is that, besides containing the connectives and quantifiers of ordinary logic, terms can be of the form $\lambda y.P$, meaning that the variable y is bound in the body P. Also the notion of application as higher-order term passing is already embedded in the logic. The logic is equipped with a sequent calculus that takes care of this λ-calculus style features and also deals with the technicalities that arise with dealing with binders. Another distinctive feature of the logic $FO\lambda^{\Delta\nabla}$ is the use of the *new quantifier* ∇. Basically the meaning of a formula $\nabla x.\Phi$ is that x is a fresh new name within the scope of the quantifier ∇, i.e. within the formula Φ.

A user can use the logic in order to specify the semantics of calculi with binders. In [93], the authors formulate the late and early π-calculus. The semantics of these calculi are presented conveniently by a set of rules. For instance, the rule

$$\frac{P \xrightarrow{\alpha} R}{P + Q \xrightarrow{\alpha} R}$$

is expressed by the $FO\lambda^{\Delta\nabla}$ formula $(P \xrightarrow{\alpha} R) \supset P + Q \xrightarrow{\alpha} R$. This formula formalizes a part of the behaviour of the choice operator in much the same way as the rule (sum1) does in our formulation of Section 5.5.2. The symbol $\supset$ stands for the standard implication, P,Q and R are variables ranging over terms, $\xrightarrow{\alpha}$ is a binary relation symbol[10] in the signature of the logic and + is a binary function symbol in the signature of the logic. The whole set of rules that define the late and the early π-calculus can be found in [93]. Here we discuss only a few rules that highlight the characteristic features of the approach. Consider for instance the following rule

$$\frac{\nabla x.(Px \xrightarrow{A} Qx)}{\nu x.(Px) \xrightarrow{A} \nu x.(Qx)} \text{ (parRes1)}$$

Thanks to the quantifier ∇, the premise of the rule is satisfied when the process that instanciates the variable P performs a transition when the name x in its body is chosen fresh. If this happens then the term $\nu x.P$ can progress accordingly. The quantifier ∇ is thus very expressive, and offers the possibility to treat names as

[10] In [93], relations also are labeled with a type. In order to ease the presentation we here omit type information.

fresh new ones when needed. This feature turns out to be frequently useful, especially in the context of the π-calculus and similar calculi.

The reader may want to consider also the rule for the input prefix

$$\frac{}{in(\lambda y.M) \overset{\downarrow XU}{\rightarrow} (MU)} \text{ (in)}$$

In this rule X and U represent variables for channel names. The computational step for input prefix processes is formalized by means of the λ-calculus style parameter passing embedded in the logic.

The sequent calculus for $FO\lambda^{\Delta\nabla}$ takes care of deriving the expected formulae, dealing with the binders with its λ-calculus style features and with special management for the quantifier ∇. The sequent calculus is not showed here, the interested reader can consult [93] and [92] for it.

The main difference between Nominal SOS and the approach in [93] is that in Nominal SOS no technicality concerning the use of binders is hidden and given as built-in. Moreover, Nominal SOS does not rely on existing notions for parameter passing. The language specifier needs to define nearly everything. This may give the user also the possibility to explore, for instance, other types of substitution or equivalence of terms.

5.8.3 SOS in Abella

In [64], Gacek, Miller and Nadathur describe a method to specify calculi with binders. They make use of the λ-tree syntax approach, discussed in the previous section about the approach with $FO\lambda^{\Delta\nabla}$. The aim of the authors is to use the proof assistant Abella, [62], in order to reason about calculi with binders. Abella is a proof assistant for the specification logic G, [63], which we do not discuss here. In the approach of [64], the authors do not make use of the logic G directly; they instead provide a second specification logic that is more suited for their purposes. This latter logic is the theory of the intuitionistic second-order hereditary Harrop formulae, there called hH^2. The logic hH^2 turns out to be a convenient vehicle in order to formulate rule-based definitions such as the ones encountered in SOS. It also turns out that, since hH^2 is a subset of $\lambda Prolog$, [111], specifications in hH^2 can be computed and executed effectively.

The authors provide an encoding from formulae of hH^2 into the logic G, making the use of the Abella proof assistant possible.

In [64], the authors formalize the simply typed call-by-value λ-calculus, see [34]. Roughly speaking, in the call-by-value strategy, the λ-calculus parameter passing involved in the application $(\lambda x.M)\ N$ is performed only when the term N is a value, i.e. when N is an abstraction. The term N is thus first evaluated until it becomes a value. For the sake of the presentation, we show the code for the call-by-value λ-calculus in its untyped version.

∀ a,r [value(abs a r)]

∀ m,n,m′ [step m m′ ⊃ step(app m n) step(app m′ n)]

∀ m,n,n′ [value m ∧ step n n′ ⊃ step(app m n) step(app m n′)]

∀ a, r, m [value m ⊃ step(app(abs a r) m) (r m)]

The predicate 'value' recognizes values, i.e. abstractions. The term $(app\ m\ n)$ corresponds to the application $(M\ N)$. In the formulation of [64], the term $(abs\ a\ (\lambda x.r\ x))$ represents an abstraction in the programming language in which the bound variable x has type a. The predicate 'step' is defined by the last three implications in the expected way.

Since the theory proposed in [64] follows the λ-tree approach, its differences with Nominal SOS are by and large the same stated in the previous section, concerning the approach with $FO\lambda^{\Delta\nabla}$. As another difference, we can see that the logic hH^2 lacks an explicit and clear mechanism for stating freshness conditions. Note, however, that the specification logic G of Abella is in general more powerful and, in particular, it is capable of expressing freshness conditions. When necessary, the language specifier can modify the code produced by the encoding from hH^2 into the logic G in order to state freshness conditions.

5.8.4 SOS with Ott

In [135], Sewell et al. describe Ott, a tool support for the specification of calculi with binders in SOS. The general idea behind Ott is similar to the one for the approach described in the previous section for SOS specifications in Abella. The authors provide a meta language for formulating the semantics of calculi with binders and the purpose of the Ott tool support is to generate from the specification the code for some popular proof assistants. In this approach, the proposed meta-

language is not a logic as in the case of the previous section; it is instead a simple and intuitive language for which we shall give an example below.

Again, we do not describe the entire system, but the reader can obtain an idea of how this system works by means of an example. In [135], the authors formalize the call-by-value λ-calculus. As the language tends to be particularly verbose, below we present our formulation of the simpler call-by-name λ-calculus. The code below has been also stripped of some decorating syntax and those parts that are not relevant to understand Ott at this stage. The interested reader is invited to read [135] for a full description of the system and the syntax employed.

```
grammar
term t :: 't_' ::=
        x                ::       var
        \x . t           ::       lam (+ bind x in t +)
        t t'             ::       app

terminals :: 'terminals_' ::=
        \        ::       lambda {{tex \lambda}}
(1)     -->      ::       red {{tex \longrightarrow}}

defn
t_1 --> t_2 ::reduce:: {{com [[t_1]] reduces to [[t_2]] }}


------------------------ :: ax
\x . t_1 t_2 --> {t_2/x}t_1

t_1 --> t_1'
------------------------ :: rule1
t_1 t --> t_1' t
```

The first part of the code is devoted to the formalization of the terms of the calculus. It is important to note that the language provided by Ott requires the user to use the special decoration *(+ bind x in t +)* in order to specify binding information. The rest is quite simple to understand. In particular, the language provided by Ott allows the user to specify the SOS semantic rules for the calculus at hand as she/he would *draw* them from the paper to the text editor.

The tool support Ott takes the specification and returns code for the most popular interactive theorem provers, such as Coq [23], HOL [122] and Isabelle/HOL [114]. Nicely, Ott also returns code for a LaTeX presentation of the calculus. The reader can see that the listing above is indeed disseminated every now and then with presentational information. For instance, with the line that we labelled (1), the latex code generated will be such that the transition relation denoted $--{>}$ in the text editor will be represent by the symbol $\longrightarrow$.

Nominal SOS and the approaches discussed in the previous sections employ a single binding. Roughly speaking, they are abstractions in the style of λ-calculus where in the term $\lambda x.M$ only one variable can be bound. Ott allows for more complex binding structures, for instance structured patterns, multiple mutually recursive let definitions, and dependent record patterns, see [135] and [69].

However, the single binding employed by Nominal SOS and by the other frameworks has proved to be expressive enough for most of the cases.

5.8.5 General Remarks

As a general remark on the related work, the mentioned systems attack the problem of defining the operational semantics of nominal calculi using an approach that is different from the one proposed in this chapter. Our long-term goal is to develop systematically a meta-theory of SOS for calculi with binders and, following the lead of the meta-theory of ordinary SOS, to investigate at a syntactic level those semantic phenomena that are specifically concerned with binding.

From this point of view, the previous approaches do not seem close enough to the standard framework of ordinary SOS, while Nominal SOS, being a slight variation of it with specific primitive notions for dealing with binders, is fairly close. We believe that the close relationship between Nominal SOS and ordinary SOS will have the following benefits.

1. We will be able to lift/adapt already existing meta-theory to the context of binders with relative little effort. Transporting the body of meta-theoretic results to the contexts of the other approaches seems to be a more difficult task.

2. The content of the meta-theorems and the line of investigation will resemble closely those achieved so far for ordinary SOS, and with which SOS users are familiar.

3. Moreover, thanks to the nominal approach, we hopefully have an intuitive and familiar language with which we could explain why certain calculi afford a property while others do not, for instance by means of presence/absence of freshness premises, or of a syntactic discipline in the use of them or of other related nominal concepts.

Carrying out a study of the meta-theory of languages with binders based on the other approaches is still possible, but it is not part of our immediate future research goals.

5.9 Conclusions and Future works

In this chapter, we have introduced a framework, called Nominal SOS, for modelling the operational semantics of nominal calculi. The framework comes equipped with the basic features used in defining such calculi, namely, substitution and α-conversion. Since these notions are generated from the nominal signature they can be replaced by, and the user can experiment with, for instance, other notions of substitution. Our syntax has two levels of variables like the one in [145], but we do not take permutations as primitive—because there is no need to do so; we obtain α-conversion 'for free' based on atom-for-atom substitutions as a nominal SOS theory. This suffices for our examples of interest, e.g. the λ- and π-calculi.

We used the framework to specify the semantics of the lazy λ-calculus and early π-calculus and showed that our formulations of the semantics coincide with the original ones. A notion of nominal bisimilarity arises naturally from our framework. Moreover, we showed that the notion of nominal bisimilarity in our semantics of the early π-calculus coincides with open bisimilarity in the original semantics.

Nominal SOS provides a framework to extend the meta-theory of SOS to the nominal setting. This chapter contains only the basic developments of the framework, accompanied by some examples. The framework seems close enough to the formalization of the standard theory of SOS in such a way that the mode of operation for carrying out meta-theory over Nominal SOS and the content of the corresponding meta-theorems would resemble very closely those presented for instance in [18, 109]. Hence, lifting previous results from the meta-theory of standard SOS to the new context seems also feasible. Our main aim is to develop the

theory and applications of Nominal SOS so that it reaches a level of maturity that is comparable to that of the theory of classic SOS, as surveyed in, e.g., [18, 109]. More specifically, the main goals of our future work are as follows.

- We intend to provide further evidence that Nominal SOS is expressive enough to capture the original semantics of nominal calculi, such as variants of the π-calculus and its higher-order version [130], the psi-calculi [36] and the object calculi [1], and to prove formally the correspondence between the presentation in terms of Nominal SOS and the original ones. Also, we plan to address different notions of equivalence betweens terms. Just to name a few examples, it would be worth defining within Nominal SOS a notion that is the analogous of the applicative bisimilarity in the context of the λ-calculus, [2], and investigating the relation between the notion of nominal bisimilarity and the applicative bisimilarity. Also, an adaptation of the nominal bisimilarity that coincides with the open bisimilarity with distinctions, [132, 131], when applied to π-calculus terms.

- We plan to develop the meta-theory of Nominal SOS, for example by providing congruence formats ifor behavioural semantics n the context of calculi with binders, possibly generalizing those proposed in [153] and [56], for instance. Also, *confluence* is an important property that has not been tackled yet by the theory of SOS in the context of rule formats. Many important confluence results stem from the realm of calculi equipped with binders; the reader may indeed think of the λ-calculus and its variants. It would be thus desirable to provide rule formats guaranteeing the confluence property within the framework of Nominal SOS. Meta-theory over Nominal SOS can be carried out also for all those phenomena that are specifically related to binders. For instance it would be worth providing rule formats guaranteeing that the late and early bisimilarity, see [132], coincide.

- We expect to extend a wealth of classic SOS meta-results and techniques to the framework of Nominal SOS.

- We plan on providing tool support for Nominal SOS.

5.9.1 Extensions of the framework

In a sense, in this chapter we have formulated Nominal SOS in its most basic form. We are aware of a few possible extensions of the framework, which would be worth adding.

Negative freshness premises. A possible extension of the framework would be to add the possibility to have premises of rules of the form $\neg(a\#t)$, with t a term. Intuitively, this premise is satisfied when the atom a is *not* fresh in the term t. Negative freshness tests as premises do not seem to be employed in general in the definition of nominal calculi, and thus they are left out in our formulation of Section 5.3.

Variables in bound terms. It would be worth extending the syntax of nominal terms of Section 5.2 by augmenting the grammar with the following form of terms

$$t \quad ::= \quad \ldots \mid ([x_{\mathbb{A}}]t_{\sigma})_{[\mathbb{A}]\sigma}$$

Intuitively, x is a variable of atom sort. The reader may recall that the terms defined in Section 5.2 are such that abstractions are only of form $[a].t$ with a concrete atom a. It is important to notice that, given a term t, the term $[x].t$ is an open term while the term $[a].t$ is closed. This addition should be accompanied by an extension of the set of possible premises in rules so that also premises of the form $x\#t$, with x a variable of atom sort are allowed. By way of example, these two extensions would allow Nominal SOS to formulate the semantics of the lazy λ-calculus with the following two rules.

$$\frac{}{\lambda([z]x) \to \lambda([z]x)} \qquad \frac{x_0 \to \lambda([z]y_0) \quad y_0 \overset{z \overset{T}{\mapsto} x_1}{\longrightarrow} y_1 \quad y_1 \to y_2}{x_0\ x_1 \to y_2}$$

The rules above use the variable z in order to range over the set of atoms. These two rules alone can replace the rules (abs) and (app) of Section 5.5.1, which use concrete atoms and they are thus replicated for every atom. The semantics of the λ-calculus would be thus finitely formalized.

Specific syntax to state the meaning of binders. Another possible extension of the Nominal SOS framework is to augment the signature of an NTSS with information that concerns the meaning of the binders in the language. The meaning of binders used in nominal calculi can indeed be of various nature. For instance, the binder λ in the formulation of the λ-calculus binds its argument atom in a way that supports a substitution for it in a parameter passing fashion. On the other hand, the meaning of the binder ν in the formulation of the π-calculus binds its argument atom to indicate that its name must be considered private.

It would be worth investigating a suitable language that allows the user to express how binders are intended to be used in the calculus. Insofar this subject is concerned, we have no preliminary developments so far.

5.10 Useful definitions for nominal terms.

We recall the syntax for nominal terms given in Definition 5.2.2.

$$t \quad ::= \quad x_\sigma \mid a_{\mathbb{A}} \mid ([a_{\mathbb{A}}]t_\sigma)_{[\mathbb{A}]\sigma} \mid (f_{(\sigma_1,\ldots,\sigma_n)\to\delta}(t_{\sigma_1},\ldots,t_{\sigma_n}))_\delta$$

We define $vars(t)$, the set of variables that occur in the term t, by induction on the structure of t.

$$\begin{aligned} vars(x) &= \{x\} \\ vars(a) &= \emptyset \\ vars([a]t) &= vars(t) \\ vars(f(t_1,\ldots,t_n)) &= vars(t_1) \cup \ldots \cup vars(t_n) \end{aligned}$$

We define $\mathcal{A}(t)$, the set of atoms that occur in the term t, by induction on the structure of t.

$$\begin{aligned} \mathcal{A}(x) &= \emptyset \\ \mathcal{A}(a) &= \{a\} \\ \mathcal{A}([a]t) &= \{a\} \cup \mathcal{A}(t) \\ \mathcal{A}(f(t_1,\ldots,t_n)) &= \mathcal{A}(t_1) \cup \ldots \cup \mathcal{A}(t_n) \end{aligned}$$

We define $ba(t)$, the set of bound atoms that occur in the term t, by induction on the structure of t.

$$\begin{aligned} ba(x) &= \emptyset \\ ba(a) &= \emptyset \\ ba([a]t) &= \{a\} \cup ba(t) \\ ba(f(t_1, \ldots, t_n)) &= ba(t_1) \cup \ldots \cup ba(t_n) \end{aligned}$$

We define $fa(t)$, the set of atoms a in $\mathcal{A}(t)$ that have an occurrence in t that is not within the scope of an abstraction $[a]\_$, by induction on the structure of t.

$$\begin{aligned} fa(x) &= \emptyset \\ fa(a) &= \{a\} \\ fa([a]t) &= fa(t) - \{a\} \\ fa(f(t_1, \ldots, t_n)) &= fa(t_1) \cup \ldots \cup fa(t_n) \end{aligned}$$

5.11 Useful definitions for λ-terms.

We define $FV(M)$, the set of free variables that occur in a λ-term M, by induction on the structure of M.

$$\begin{aligned} FV(a) &= a \\ FV(\lambda a.M) &= FV(M) - \{a\} \\ FV((M\ N)) &= FV(M) \cup FV(N) \end{aligned}$$

We say that a λ-term M is *closed* if $FV(M) = \emptyset$, i.e., if M does not contain free variables.

Given a variable a and two λ-terms M and N, we define the *substitution* operation $M[N/a]$ defined inductively on the structure of M. In what follows a and b are considered different variables.

$$\begin{aligned} a[N/a] &= N \\ b[N/a] &= b \\ (\lambda a.M)[N/a] &= \lambda a.M \\ (\lambda b.M)[N/a] &= \lambda a.(M[N/a]),\ if\ b \notin FV(N) \\ (M_1\ M_2)[N/a] &= (M_1[N/a]\ M_2[N/a]) \end{aligned}$$

We also recall that *α-equivalence* in λ-calculus is defined as the least congruence $\equiv$ on λ-terms such that if $y \notin FV(M)$ then $\lambda x.M \equiv \lambda y.(M[y/x])$.

5.12 Useful definitions for π-terms.

We here recall that a label α can be of the following form

$$\alpha \quad ::= \quad \tau \mid \overline{a}b \mid ab \mid \overline{a}(b)$$

where a and b are two channel names.

We define $names(\alpha)$, the set of names that occur in the label α, as follows.

$$\begin{aligned} names(\overline{a}b) &= \{a, b\} \\ names(ab) &= \{a, b\} \\ names(\overline{a}(b)) &= \{a, b\} \end{aligned}$$

We define $bn(\alpha)$, the set of names in the label α that are bound, as follows.

$$\begin{aligned} bn(\overline{a}b) &= \emptyset \\ bn(ab) &= \emptyset \\ bn(\overline{a}(b)) &= \{b\} \end{aligned}$$

We define $fn(P)$, the set of names that occur free in the process P, as follows.

$$\begin{aligned} fn(\mathbf{0}) &= \emptyset \\ fn(\tau.P) &= fn(P) \\ fn(\overline{a}b.P) &= \{a, b\} \cup fn(P) \\ fn(a(b).P) &= \{a\} \cup fn(P) - \{b\} \\ fn(va.P) &= fn(P) \; - \; \{a\} \\ fn(P + Q) &= fn(P) \cup fn(Q) \\ fn(P \parallel Q) &= fn(P) \cup fn(Q) \\ fn(!P) &= fn(P) \end{aligned}$$

Given a π-term P and distinct names a and b, we define the *substitution* operation $P[b/a]$ inductively on the structure of P. In what follows a, b, c and d ranges over names. The substitution of names acts in the expected way. In particular $a[b/a] = b$ and $c[b/a] = c$ when $c \neq a$.

$$\begin{aligned} \mathbf{0}[b/a] &= \mathbf{0} \\ (\tau.P)[b/a] &= \tau.(P[b/a]) \\ (\overline{c}d.P)[b/a] &= (\overline{c[b/a]}d[b/a].P[b/a]) \\ (c(d).P)[b/a] &= c[b/a](d).(P[b/a])) \quad \textit{with } d \neq a \\ (b(a).P)[b/a] &= b(a).P \\ (vc.P)[b/a] &= vc.(P[b/a]) \quad \textit{with } c \neq a \\ (va.P)[b/a] &= va.P \\ (P + Q)[b/a] &= P[b/a] + Q[b/a] \\ (P \parallel Q)[b/a] &= P[b/a] \parallel Q[b/a] \\ (!P)[b/a] &= !(P[b/a]) \end{aligned}$$

We also recall that *α-equivalence* in π-calculus is defined as the least congruence $\equiv$ on π-terms such that

- if $c \notin fn(a(b).P)$ then $a(b).P \equiv a(c).(P[c/b])$.
- if $b \notin fn(P)$ then $va.P \equiv vb.(P[b/a])$.

5.13 Correctness of Substitution Transitions w.r.t. Syntactic Substitution: Proof of Theorem 5.4.2

We prove the two implications in the statement of Theorem 5.4.2 separately.

- $\Rightarrow$: $M \xrightarrow{a \overset{T}{\mapsto} N} M[N/a]$
- $\Leftarrow$: if $M' = M[N/a]$ then $M \xrightarrow{a \overset{T}{\mapsto} N} M'$

$\Rightarrow$

Let T be a *NTSS* and let us pick up closed terms M and N from the signature of T and an atom a. Suppose that $M \xrightarrow{a \overset{T}{\mapsto} N} M'$. We prove that $M' = M[N/a]$.

The proof is by induction on the structure of the term M.

- Case $M = a$: There is only one possible substitution transition, $a \xrightarrow{a \overset{T}{\mapsto} N} N$ by rule (a1_{Ts}). Indeed, $a[N/a] = N$.
- b, with $b \neq a$: There is only one possible substitution transition, $b \xrightarrow{a \overset{T}{\mapsto} N} b$, by rule ($\text{a2}_{\text{Ts}}$): Indeed, when b is different from a, we have that $\text{b}[N/a] = b$.
- Case $M = [a]M_1$: There is only one possible substitution transition, $[a]M_1 \xrightarrow{a \overset{T}{\mapsto} N} [a]M_1$ by rule (abs2_{Ts}). Indeed $([a]M_1)[N/a] = [a]M_1$.
- Case $M = [b]M_1$, with $b \neq a$: There is only one possible substitution transition, $[b]M_1 \xrightarrow{a \overset{T}{\mapsto} N} [b]M'$ by rule (abs1_{Ts}): Since this transition is provable, we have that $b\#N$ and $M_1 \xrightarrow{a \overset{T}{\mapsto} N} M'$. By the inductive hypothesis, which applies to M_1, we have that $M_1 \xrightarrow{a \overset{T}{\mapsto} N} M_1[N/a]$. Having b fresh in N and $b \neq a$ we have indeed $([b]M_1)[N/a] = [b](M_1[N/a])$.
- Case $M = f(M_1, M_2, \ldots, M_n)$: The only possible transitions are with rule (f_{Ts}). Assume $f(M_1, M_2, \ldots, M_n) \xrightarrow{a \overset{T}{\mapsto} N} f(M'_1, M'_2, \ldots, M'_n)$. Since this transition

is provable, we have that $M_1 \stackrel{a \stackrel{T}{\mapsto} N}{\longrightarrow} M'_1$, $M_2 \stackrel{a \stackrel{T}{\mapsto} N}{\longrightarrow} M'_2$, $\ldots$, $M_n \stackrel{a \stackrel{T}{\mapsto} N}{\longrightarrow} M'_n$. By the inductive hypothesis, which applies to M_1, M_2, $\ldots$ and M_n, we know that $M'_i = M_i[N/a]$ for each $i \in \{1, \ldots, n\}$.

So $f(M_1, M_2, \ldots, M_n) \stackrel{a \stackrel{T}{\mapsto} N}{\longrightarrow} f(M_1[N/a], M_2[N/a], \ldots, M_n[N/a])$.

Indeed $f(M_1, M_2, \ldots, M_n)[N/a] = f(M_1[N/a], M_2[N/a], \ldots, M_n[N/a])$.

$\Leftarrow$

Suppose that $M' = M[N/a]$. We wish to prove that $M \stackrel{a \stackrel{T}{\mapsto} N}{\longrightarrow} M'$ is a provable transition. This can be proved by induction on the structure of M, following the lines of the proof above.

5.14 Correctness of α-Conversion Transitions w.r.t. Syntactic α-Conversion: Proof of Theorem 5.4.4

The proof of Theorem 5.4.4 relies on the correctness of atom-for-atom substitution transitions with respect to the syntactic substitution. We shall state this correctness theorem explicit.

Theorem 5.14.1 (Correctness of atom-for-atom Substitution Transitions) *Let T be an NTSS and let M be a closed terms from the signature of T. Then, for all atoms a and b, and for each term M' it holds that $M \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} M'$ if and only if $M' = M[b/a]$.*

The proof of Theorem 5.14.1 follows exactly the same line as in the proof of Theorem 5.4.2 (correctness of term-for-atom substitutions) in Section 5.13, and it is omitted.

The following lemmas will be necessary to complete the proof. Their proofs are straightforward and thus omitted.

Lemma 5.14.2 (Freshness after substitution.) *Let T be an NTSS and let M be a closed term from the signature of T. Then, for all atoms a and b it holds that $a\#M[b/a]$.*

Lemma 5.14.3 (Substitution is involutive.) *Let T be an NTSS and let M be a closed term from the signature of T. Then, for all atoms a and b it holds that if $b\#M$ then $M = (M[b/a])[a/b]$.*

Lemma 5.14.4 (Existence of a substitution.) *Let T be an NTSS and let M be a closed term from the signature of T. Then, for all atoms b it holds that if $b\#M$ then the transition $M \xrightarrow{a \overset{A}{\mapsto} b} M'$ is provable, for some M'.*

We prove the two implications in the statement of Theorem 5.4.4 separately.

- $\Rightarrow$: if $M \to_\alpha N$ then $M =_\alpha N$.
- $\Leftarrow$: if $M =_\alpha N$ then $M \to_\alpha N$.

$\Rightarrow$

Let T be an NTSS and assume that $M \to_\alpha N$. We prove that $M =_\alpha N$. The proof is by induction on the length of the proof of the transition $M \to_\alpha N$.

- Proofs of length 1: The only provable transition with length 1 is by rule (id_α), $M \to_\alpha M$. In this case, by reflexivity of $=_\alpha$, we have that $M =_\alpha M$.
- Proofs of length $n > 1$: We proceed by a case analysis on the last rule used in the proof of the transition $M \to_\alpha N$.
  - Suppose that $M = [a]M_1 \to_\alpha [b]M' = N$ using rule ($\mathrm{abs1}_\alpha$), with $b\#M_1$ and $M_1 \xrightarrow{a \overset{A}{\mapsto} b} M'$. By Theorem 5.14.1, which states the correctness of the substitution transitions, we know that $M' = M_1[b/a]$. The transition actually proved is thus $[a]M_1 \to_\alpha [b](M_1[b/a])$. Since b is fresh in M_1, we have indeed that $[a]M_1 =_\alpha [b](M_1[b/a])$.
  - Suppose that $M = [a]M_1 \to_\alpha [a]M' = N$ using rule ($\mathrm{abs2}_\alpha$), with $M_1 \to_\alpha M'$. Since the proof length for proving the transition $M_1 \to_\alpha M'$ is strictly less than n, by the inductive hypothesis we can conclude that $M_1 =_\alpha M'$. Since $=_\alpha$ is a congruence, we can place M_1 and M' in a same context, thus $[a]M_1 =_\alpha [a]M' = N$.
  - Suppose that $M = f(M_1, M_2, \ldots, M_n) \to_\alpha f(M_1', M_2', \ldots, M_n') = N$ using rule (f_α), with $M_1 \to_\alpha M_1'$, $M_2 \to_\alpha M_2'$, $\ldots$, $M_n \to_\alpha M_n'$. Since the proof length for proving all the mentioned transitions is strictly less than n, by the inductive hypothesis we can conclude that $M_1 =_\alpha M_1'$, $M_2 =_\alpha M_2'$, $\ldots$, $M_n =_\alpha M_n'$. As in the previous case, since $=_\alpha$ is a congruence, $f(M_1, M_2, \ldots, M_n) =_\alpha f(M_1', M_2', \ldots, M_n') = N$.
  - Suppose that $M = M_1 \to_\alpha M_3 = N$ using rule $\alpha \cdot \mathrm{upTo}\alpha$, with $M_1 \to_\alpha M_2$ and $M_2 \to_\alpha M_3$, for some closed term M_2. Since the proof length for

proving the transitions $M_1 \to_\alpha M_2$ and $M_2 \to_\alpha M_3$ are both strictly less than n, by the inductive hypothesis we can conclude that $M_1 =_\alpha M_2$ and $M_2 =_\alpha M_3$. Since $=_\alpha$ is a congruence, it is a transitive relation, thus $M_1 =_\alpha M_3 = N$.

$\Leftarrow$

Assuming the hypothesis, let T be a $NTSS$. The proof is by induction on the lenght of the proof of $M =_\alpha N$. We proceed by a case analysis on the last rule used in the proof.

- Length 1: There are two types of trees with length 1.
  - Suppose that $M =_\alpha M$ by reflexivity. We can use the rule (id_α) in order to prove $M \to_\alpha M$.
  - Suppose that $[a]M =_\alpha [b](M[b/a])$, with b fresh in M. This is the base case of the syntactic α-conversion. By Theorem 5.14.1, which states the correctness of the substitution transitions, we know that $M \xrightarrow{a \mapsto b} M[b/a]$. We can use the rule ($\mathrm{abs1}_\alpha$) instanciated in the following way.

$$\frac{M \xrightarrow{a \mapsto b} M[b/a] \qquad b \# M}{[a]M \to_\alpha [b]M[b/a]}$$

    Since b is fresh in M, the freshness premise $b\#M$ is satisfied and we can prove $[a]M \to_\alpha [b](M[b/a])$.

- Length $n > 1$: There are three types of trees with length n, which are the inductive cases.
  - Suppose that $M =_\alpha N$ by symmetry, with $N =_\alpha M$ by a shorter proof. Since the length of the tree for deriving the equation $N =_\alpha M$ is strictly less than n, by the inductive hypothesis we can conclude that $N \to_\alpha M$. Now we have to prove that also the transition $M \to_\alpha N$ is provable. A case analysis on the last rule used in order to prove the transition $N \to_\alpha M$ shows that we can use the same rule to prove $M \to_\alpha N$. The only non-symmetric rule is ($\mathrm{abs1}_\alpha$), and we will therefore show only this case in detail.

    Consider $N = [a]M_1$ and $M = [b]M_1'$, the provable transition is thus $[a]M_1 \to_\alpha [b]M_1'$ using rule ($\mathrm{abs1}_\alpha$), with $b\#M_1$ and $M_1 \xrightarrow{a \mapsto b} M_1'$. By

Theorem 5.14.1, which states the correctness of the substitutions, we know that $M_1' = M_1[b/a]$. The transition actually proved is thus $[a]M_1 \rightarrow_\alpha [b]M_1[b/a]$. We thus can instanciate the rule $(abs1_\alpha)$ as follows.

$$\frac{M_1[b/a] \xrightarrow{b \mapsto a} (M_1[b/a])[a/b] \qquad a\#M_1[b/a]}{[b](M_1[b/a]) \rightarrow_\alpha [a](M_1[b/a][a/b])}$$

By Lemma 5.14.2 we have that $a\#M_1[b/a]$. The freshness premise $a\#M_1[b/a]$ is thus satisfied. Since $a\#M_1[b/a]$, by Lemma 5.14.4 we can conclude that the transition $M_1[b/a] \xrightarrow{b \mapsto a} M'$ is provable, for some M'. Moreover, Theorem 5.14.1, which states the correctness of the substitutions, guarantees that this particular $M' = M_1[b/a][a/b]$ (the theorem is applied to the term $M_1[b/a]$).

Since $b\#M_1$, by Lemma 5.14.3 we have that $M_1[b/a][a/b] = M_1$, and we thus are able to prove the transition $[b]M_1[b/a] \rightarrow_\alpha [a]M_1$, i.e. $M \rightarrow_\alpha N$.

– Suppose that $M_1 =_\alpha M_3$ by transitivity, with $M_1 =_\alpha M_2$ and $M_2 =_\alpha M_3$. Since the length of the trees for deriving the equations $M_1 =_\alpha M_2$ and $M_2 =_\alpha M_3$ are both strictly less than n, by inductive hypothesis we can conclude that $M_1 \rightarrow_\alpha M_2$ and $M_2 \rightarrow_\alpha M_3$. We can thus use the rule $(\alpha \cdot \text{upTo}\alpha)$ as follows

$$\frac{M_1 \rightarrow_\alpha M_2 \quad M_2 \rightarrow_\alpha M_3}{M_1 \rightarrow_\alpha M_3}$$

in order to prove $M_1 \rightarrow_\alpha M_3$.

– Suppose that $M =_\alpha N$ by congruence. The reader should see that the rules for generating α-conversion transitions $\rightarrow_\alpha$ are designed to share $\rightarrow_\alpha$ in all the contexts. For the sake of example, we here show in detail only the case of function symbols.

Let us consider the equation $f(M_1, M_2, \ldots, M_n) =_\alpha f(M_1', M_2', \ldots, M_n')$, with $M_1 =_\alpha M_1', M_2 =_\alpha M_2', \ldots, M_n =_\alpha M_n'$. Since the length of the proofs for the equations $M_i =_\alpha M_i'$ $(i \in \{1, \ldots, n\})$ is strictly less than n, by the inductive hypothesis we can conclude that $M_1 \rightarrow_\alpha M_1'$, $M_2 \rightarrow_\alpha M_2'$, $\ldots$, $M_n \rightarrow_\alpha M_n'$. We can thus instanciate the rule (f_α) in the following way

$$\frac{M_1 \rightarrow_\alpha M_1' \quad M_2 \rightarrow_\alpha M_2' \quad \cdots \quad M_n \rightarrow_\alpha M_n'}{f(M_1, M_2, \ldots, M_n) \rightarrow_\alpha f(M_1', M_2', \ldots, M_n')}$$

in order to prove $f(M_1, M_2, \ldots, M_n) \rightarrow_\alpha f(M_1', M_2', \ldots, M_n')$.

5.15 Correctness of λ: Proof of Theorem 5.5.1

The encoding $[\![\cdot]\!]^\lambda$ is the map from Λ into terms of the nominal λ-calculus defined in Section 5.5.1, here repeated.

$$\begin{array}{rcl} [\![a]\!]^\lambda & = & a \\ [\![\lambda a.M]\!]^\lambda & = & \lambda([a][\![M]\!]^\lambda) \\ [\![(M\ N)]\!]^\lambda & = & ([\![M]\!]^\lambda\ [\![N]\!]^\lambda) \end{array}$$

Our proof of Theorem 5.5.1 relies on the following lemmas, stating the correctness of substitution and α-conversion transitions.

Lemma 5.15.1 (Correctness of substitution transitions)

$$[\![M]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} M' \text{ if and only if } M' = [\![M[N/a]]\!]^\lambda.$$

We use $\equiv$ to denote syntactic equality up to α-equivalence of the lazy λ-calculus.

Lemma 5.15.2 (Correctness of α-conversion transitions)

$$[\![M]\!]^\lambda \rightarrow_\alpha N' \text{ if and only if } M \equiv N \wedge [\![N]\!]^\lambda = N'.$$

The proofs of Lemma 5.15.1 and 5.15.2 are lengthy even though they follow standard reasoning by induction. They can be found in Section 5.17 and Section 5.19, repectively.

Here we recall the rules that define the lazy λ-calculus from [2].

$$\frac{}{\lambda a.x \rightarrow \lambda a.x}\ (\text{absO}) \qquad \frac{x_0 \rightarrow \lambda a.y_0 \quad y_0[x_1/a] \rightarrow y_1}{(x_0\ x_1) \rightarrow y_1}\ (\text{appO})$$

Also, λ-terms are considered up to α-equivalence. The substitution and α-equivalence of λ-calculus are standard and not provided here. However, for completeness of reference, they are repeated in Section 5.11.

We prove the two implications in the statement of Theorem 5.5.1 separately.

- $\Rightarrow$: if $M \to N$ then $[\![M]\!]^\lambda \to [\![N]\!]^\lambda$.
- $\Leftarrow$: if $[\![M]\!]^\lambda \to N$ then $M \to N'$ and $[\![N']\!]^\lambda = N$, for some term N'.

$\Rightarrow$

Notice that rule (appO) and consequently rule (app) of Section 5.5.1 uses a form of what in the terminology of SOS is called a *look-ahead*. The ordinary induction on the structure of terms thus fails and we below employ an induction on the length of the proofs of provable transitions in the original λ-calculus.

- Proofs of length 1: The only proofs of length 1 are with rule (absO). Given a term M, we can prove the transition $\lambda a.M \to \lambda a.M$. Now, $[\![\lambda a.M]\!]^\lambda = \lambda([a][\![M]\!]^\lambda)$. By means of rule (abs) we can prove indeed the transition $\lambda([a][\![M]\!]^\lambda) \to \lambda([a][\![M]\!]^\lambda)$, and we are done.
- Proofs of length $n > 1$: The only proofs of length n are with rule (appO) instanciated in the following way by closed terms M_1, M_2, M_3 and N.

$$\frac{M_1 \to \lambda a.M_2 \quad M_2[N/a] \to M_3}{(M_1\ N) \to M_3}$$

Our aim is to make use of the rule (app) in order to prove a transition $[\![(M_1\ N)]\!]^\lambda \to [\![M_3]\!]^\lambda$.

Now, $[\![(M_1\ N)]\!]^\lambda = ([\![M_1]\!]^\lambda\ [\![N]\!]^\lambda)$. Since the proof length for proving the transition $M_1 \to \lambda a.M_2$ is strictly less than n, by the inductive hypothesis we can conclude that $[\![M_1]\!]^\lambda \to [\![\lambda a.M_2]\!]^\lambda$, with the target term being $\lambda([a][\![M_2]\!]^\lambda)$. The first premise of (app) is thus satisfied. By Lemma 5.15.1, which states the correctness of substitution transitions, we know that $[\![M_2]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M_2[N/a]]\!]^\lambda$, the second premise of (app) is thus satisfied. Now, once again, the proof length for proving the transition $M_2[N/a] \to M_3$ is strictly less than n. By inductive hypothesis we can then conclude that $[\![M_2[N/a]]\!]^\lambda \to [\![M_3]\!]^\lambda$.

Rule (app) can thus be instanciated as follows

$$\frac{[\![M_1]\!]^\lambda \to [\![\lambda a.M_2]\!]^\lambda \quad [\![M_2]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M_2[N/a]]\!]^\lambda \quad [\![M_2[N/a]]\!]^\lambda \to [\![M_3]\!]^\lambda}{[\![M_1]\!]^\lambda\ [\![N]\!]^\lambda \to [\![M_3]\!]^\lambda}$$

Proving thus the transition $[\![M_1\ N]\!]^\lambda \to [\![M_3]\!]^\lambda$.

$\Leftarrow$

The proof is by induction on the length of the proofs for provable transitions in our formulation of the λ-calculus.

- Proofs of length 1: The only proofs of length 1 are with rule (abs). Given a term M, we can prove the transition $\lambda([a][\![M]\!]^\lambda) \to \lambda([a][\![M]\!]^\lambda)$. Since $[\![\lambda([a]M)]\!]^\lambda = \lambda([a].[\![M]\!]^\lambda)$ and $\lambda a.M \to \lambda a.M$, we are done.
- Proofs of length $n > 1$: There are two type of proofs of length n.
  - transitions proved with (app): Given a term M this rule is instanciated in the following way

$$\frac{[\![M_1]\!]^\lambda \to \lambda([a]M_2') \quad M_2' \overset{a \mapsto [\![N]\!]^\lambda}{\longrightarrow} M_3' \quad M_3' \to M_4'}{[\![M_1]\!]^\lambda\ [\![N]\!]^\lambda \to M_4'}$$

    Our aim is to make use of the rule (appO) in order to prove a transition $M_1\ N \to M_4$ with $[\![M_4]\!]^\lambda = M_4'$. First of all, consider the first premise of rule (app), satisfied by the transition $[\![M_1]\!]^\lambda \to \lambda([a]M_2')$. Since the length of the proof of such a transition is strictly less than n, we know that $M_1 \to M_1'$ for some M_1' such that $[\![M_1']\!]^\lambda = \lambda([a].M_2')$. It follows that $M_1' = \lambda a.M_2$ for some M_2 such that $[\![M_2]\!]^\lambda = M_2'$. Now, by Lemma 5.15.1, which states the correctness of substitution transitions, we know that $[\![M_2]\!]^\lambda \overset{a \mapsto [\![N]\!]^\lambda}{\longrightarrow} [\![M_2[N/a]]\!]^\lambda$, i.e. $M_3' = [\![M_2[N/a]]\!]^\lambda$ and since the provable transition $[\![M_2[N/a]]\!]^\lambda \to M_4'$ has proof length strictly less than n, we can conclude that $M_2[N/a] \to M_4$, with $[\![M_4]\!]^\lambda = M_4'$.

    Rule (appO) can thus be instanciated as follows

$$\frac{M_1 \to \lambda a.M_2 \quad M_2[N/a] \to M_4}{M_1\ N \to M_4}$$

    proving thus the transition $M_1\ N \to M_4$ with $[\![M_4]\!]^\lambda = M_4'$.
  - transitions proved with ($l \cdot \text{upTo}\alpha$): Given a term M this rule is instanciated in the following way

$$\frac{[\![M_1]\!]^\lambda \rightarrow_\alpha M_2' \quad M_2' \rightarrow M_3'}{[\![M_1]\!]^\lambda \rightarrow M_3'}$$

By Lemma 5.15.2, which states the correctness of α-conversion transitions, the fact that the transition $[\![M_1]\!]^\lambda \rightarrow_\alpha M_2'$ is provable, means that $M_1 \equiv M_2$ with $[\![M_2]\!]^\lambda = M_2'$. Now, since the provable transition $[\![M_2]\!]^\lambda \xrightarrow{l} M_3'$ has proof length strictly less than n, we can conclude that $M_2 \xrightarrow{l} M_3$ and so $M_1 \xrightarrow{l} M_3$ with $[\![M_3]\!]^\lambda = M_3'$.

5.16 Correctness of early π: Proof of Theorem 5.5.2

The encoding $[\![\cdot]\!]^\pi$ is a map from Π into terms of the nominal early π-calculus defined in Section 5.5.2.

$$\begin{array}{rcl}
[\![\mathbf{0}]\!]^\pi & = & \mathbf{0} \\
[\![\tau.P]\!]^\pi & = & \tau.[\![P]\!]^\pi \\
[\![\overline{a}b.P]\!]^\pi & = & out(a, b, [\![P]\!]^\pi) \\
[\![a(b).P]\!]^\pi & = & in(a, [b][\![P]\!]^\pi) \\
[\![\nu a.P]\!]^\pi & = & \nu([a][\![P]\!]^\pi) \\
[\![P + Q]\!]^\pi & = & [\![P]\!]^\pi + [\![Q]\!]^\pi \\
[\![P \parallel Q]\!]^\pi & = & [\![P]\!]^\pi \parallel [\![Q]\!]^\pi \\
[\![!P]\!]^\pi & = & ![\![P]\!]^\pi
\end{array}$$

Since we use a different notation for actions, the encoding is extended to labels as follows.

$$\begin{array}{rcl}
[\![\tau]\!]^\pi & = & \tau \\
[\![ab]\!]^\pi & = & in(a, b) \\
[\![\overline{a}b]\!]^\pi & = & out(a, b) \\
[\![\overline{a}(b)]\!]^\pi & = & bout(a, [b])
\end{array}$$

Our proof of Theorem 5.5.2 relies on the following lemmas, stating the correctness of substitution and α-conversion transitions.

Lemma 5.16.1 (Correctness of substitution transitions)

$$[\![P]\!]^\pi \xrightarrow{a \mapsto b} P' \text{ if and only if } P' = [\![P[b/a]]\!]^\pi.$$

We use $\equiv$ to denote syntactic equality up to α-equivalence of the early π-calculus.

Lemma 5.16.2 (Correctness of α-conversion transitions)

$$[\![P]\!]^{\pi} \rightarrow_{\alpha} Q' \text{ if and only if } P \equiv Q \wedge [\![Q]\!]^{\pi} = Q'.$$

The proofs of Lemmas 5.16.1 and 5.16.2 are lengthy even though they follow standard reasoning by induction. They can be found in Section 5.18 and Section 5.20, respectively.

For ease of reference, we repeat below the rules for the early π-calculus given in [132][11]. In the rules that follow, $names(\alpha)$ denotes the set of names that occur in the label α, $bn(\alpha)$ denotes the set of bound names that occur in the label α and $fn(P)$ denotes the set of names of the process P that are not bound, see [132]. For completeness of reference, the definition of these sets are repeated in Section 5.12. Although these notions are standard, Section 5.12 also repeats the definitions of substitution and α-equivalence of π-calculus.

$$\frac{}{\tau.x \xrightarrow{\tau} x}\ (\tau\text{O}) \qquad \frac{}{\overline{a}b.x \xrightarrow{\overline{a}b} x}\ (\text{outO}) \qquad \frac{}{a(b).x \xrightarrow{ac} y[c/b]}\ (\text{inO})$$

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1}\ (\text{sumO1}) \qquad bn(\alpha) \cap fn(x_2) = \emptyset\ \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \parallel x_2 \xrightarrow{\alpha} y_1 \parallel x_2}\ (\text{parO1})$$

$$\frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \parallel x_2 \xrightarrow{\tau} y_1 \parallel y_2}\ (\text{comO1})$$

$$b \notin fn(x_2)\ \frac{x_1 \xrightarrow{\overline{a}(b)} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \parallel x_2 \xrightarrow{\tau} \nu b.(y_1 \parallel y_2)}\ (\text{closeO1}) \qquad \frac{x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y \parallel !x}\ (\text{replO})$$

$$a \neq z\ \frac{x \xrightarrow{\overline{z}a} y}{\nu a.x \xrightarrow{\overline{z}(a)} y}\ (\text{openO}) \qquad c \notin names(\alpha)\ \frac{x \xrightarrow{\alpha} y}{\nu c.x \xrightarrow{\alpha} \nu c.x}\ (\text{resO})$$

For the sake of brevity, we omit the symmetric versions of rules (sumO1), (parO1), (parResO1), (comO1) and (closeO1). In what follows, these are referred to as (sumO2), (parO2), (parResO2), (comO2) and (closeO2).

The proof of Theorem 5.5.2 is divided into two cases

[11] [132] presents more rules for the replication operator for technical reasons we are not concerned with, see pages 42 and 43 in that reference.

- $\Rightarrow$: if $P \xrightarrow{\alpha} Q$ then $[\![P]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} [\![Q]\!]^\pi$
- $\Leftarrow$: if $[\![P]\!]^\pi \xrightarrow{\alpha} Q$ then $P \xrightarrow{\alpha'} Q'$ for some α' and Q' such that $[\![\alpha']\!]^\pi = \alpha$ and $[\![Q']\!]^\pi = Q$

We recall that in the statement for $\Leftarrow$ the label α does not range over substitution and α-conversion transition labels.

The following lemmas will be necessary to complete the proof. Their proofs are straightforward and thus omitted.

Lemma 5.16.3 (Interplay between the set of names and freshness (1).) *Given a channel name a and a π-calculus label α, it holds that if $a \notin names(\alpha)$ then $a \notin bn([\![\alpha]\!]^\pi)$ and the freshness assertion $a\#[\![\alpha]\!]^\pi$ is derivable.*

Lemma 5.16.4 (Interplay between the set of names and freshness (2).) *Given a channel name a and a π-calculus label α. Let α' be a label such that $\alpha' = [\![\alpha]\!]^\pi$. It holds that if $a \notin bn([\![\alpha]\!]^\pi)$ and $a\#[\![\alpha]\!]^\pi$, then $a \notin names(\alpha)$.*

Lemma 5.16.5 (Freshness in terms and their encodings.) *Given a channel name a and a π-calculus process P. If a is fresh in $[\![P]\!]^\pi$ if and only if $a \notin fn(P)$.*

$\Rightarrow$

The proof is by induction on the derivation of the π-calculus transition. We proceed by a case analysis on the last rule used in the derivation.

- $\tau.P$: The only possible transition is $\tau.P \xrightarrow{\tau} P$. Now $[\![\tau.P]\!]^\pi = \tau.[\![P]\!]^\pi$, and by rule ($\tau$) we can prove the transition $\tau.[\![P]\!]^\pi \xrightarrow{\tau} [\![P]\!]^\pi$.
- $\overline{a}b.P$: The only possible transition is $\overline{a}b.P \xrightarrow{\overline{a}b} P$. Now, $[\![\overline{a}b.P]\!]^\pi = out(a, b, [\![P]\!]^\pi)$, and by rule (out), we can prove the transition $out(a, b, [\![P]\!]^\pi) \xrightarrow{out(a,b)} [\![P]\!]^\pi$.
- $a(b).P$: The possible transitions are with labels ac for all atoms c. Let us pick an atom c and consider a transition $a(b).P \xrightarrow{ac} P[c/b]$. Now, $[\![a(b).P]\!]^\pi = in(a, [b][\![P]\!]^\pi)$. By Lemma 5.16.1, which states the correctness of substitution transitions, we know that $[\![P]\!]^\pi \xrightarrow{a \mapsto c} [\![P[c/a]]\!]^\pi$, thus we can use rule (in) to prove the transition $in(a, [b][\![P]\!]^\pi) \xrightarrow{in(a,c)} [\![P[c/a]]\!]^\pi$.
- $\nu b.P$: Let us recall the fact that $[\![\nu b.P]\!]^\pi = \nu([b][\![P]\!]^\pi)$. There are two possible transitions.

– $\nu b.P \xrightarrow{\overline{a}(b)} P'$ by rule (openO), with $P \xrightarrow{\overline{a}b} P'$ and $b \neq a$. By the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{out(a,b)} [\![P']\!]^\pi$, thus the premise in rule (open) is satisfied, and we can use this rule to prove a transition $\nu([b][\![P]\!]^\pi) \xrightarrow{bout(a,[b])} [\![P']\!]^\pi$.

– $\nu b.P \xrightarrow{\alpha} \nu b.P'$ by rule (resO), with $P \xrightarrow{\alpha} P'$ and for a generic α such that $b \notin names(\alpha)$. By inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$. By Lemma 5.16.3, since $b \notin names(\alpha)$ we have that $b \notin bn([\![\alpha]\!]^\pi)$ and $b\#[\![\alpha]\!]^\pi$, thus the premises in rule (res) are satisfied, and we can instanciate this rule in the following way

$$b \notin bn(\alpha) \quad \frac{[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi \quad b\#\alpha}{\nu([b][\![P]\!]^\pi) \xrightarrow{\alpha} \nu([b][\![P']\!]^\pi)} \text{(res)}$$

to prove a transition $\nu([b][\![P]\!]^\pi) \xrightarrow{\alpha} \nu([b][\![P']\!]^\pi)$, whose target is exactly $[\![\nu b.P']\!]^\pi$.

- $P + Q$: Consider a generic transition α. There are two possible transitions, (1) $P + Q \xrightarrow{\alpha} P'$, with $P \xrightarrow{\alpha} P'$ and (2) $P + Q \xrightarrow{\alpha} Q'$, with $Q \xrightarrow{\alpha} Q'$. We here consider only (1). Now, $[\![P + Q]\!]^\pi = [\![P]\!]^\pi + [\![Q]\!]^\pi$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$, thus the premise in rule (sum1) is satisfied, and we can use this rule to prove a transition $[\![P]\!]^\pi + [\![Q]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$. Case (2) can be proven analogously.

- $P \parallel Q$: Then $[\![P \parallel Q]\!]^\pi = [\![P]\!]^\pi \parallel [\![Q]\!]^\pi$. We distinguish four cases.

  – $\alpha \notin \{bout(a,[b]) \mid a,b \in C\}$: There are two possible transitions, (1) $P \parallel Q \xrightarrow{\alpha} P' \parallel Q$, with $P \xrightarrow{\alpha} P'$ and (2) $P \parallel Q \xrightarrow{\alpha} P \parallel Q'$, with $Q \xrightarrow{\alpha} Q'$. (We treat below separately the case for $\alpha = \tau$ with a top level communication taking place). We here consider only (1). Now, $[\![P \parallel Q]\!]^\pi = [\![P]\!]^\pi \parallel [\![Q]\!]^\pi$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$, thus the premise in rule (par1) is satisfied, and we can use this rule to prove a transition $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi \parallel [\![Q]\!]^\pi$, whose target is exactly $[\![P' \parallel Q]\!]^\pi$. Case (2) can be proven analogously.

  – $\alpha \in \{bout(a,[b]) \mid a,b \in C\}$: The only possible transition is, by rule (parO1), $P \parallel Q \xrightarrow{bout(a,[b])} P' \parallel Q$, with $P \xrightarrow{bout(a,[b])} P'$ and $b \notin fn(Q)$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi$. Moreover, since $b \notin fn(Q)$, by Lemma 5.16.5 we have that $b\#[\![Q]\!]^\pi$. The two premises of rules (par1) are therefore satisfied, and we can use this rule to prove

a transition$[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi \parallel [\![Q]\!]^\pi$, whose target is exactly $[\![P' \parallel Q]\!]^\pi$.

- Assume that $P \parallel Q \xrightarrow{\tau} P' \parallel Q'$ by rule (comO1), with $P \xrightarrow{\overline{a}b} P'$ and $Q \xrightarrow{ab} Q'$ for some atoms a and b. Since $P \xrightarrow{\overline{a}b} P'$ and $Q \xrightarrow{ab} Q'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{out(a,b)} [\![P']\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{in(a,b)} [\![Q']\!]^\pi$, thus the premises in rule (com1) are satisfied, and we can use this rule to prove a transition $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{\tau} [\![P']\!]^\pi \parallel [\![Q']\!]^\pi$, whose target is exactly $[\![P' \parallel Q']\!]^\pi$.
- Assume that $P \parallel Q \xrightarrow{\tau} \nu b.(P' \parallel Q')$ by rule (closeO1), with $P \xrightarrow{\overline{a}(b)} P'$ and $Q \xrightarrow{ab} Q'$ for some atoms a and b, with $b \notin fn(Q)$. Now, since $P \xrightarrow{\overline{a}(b)} P'$ and $Q \xrightarrow{ab} Q'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{in(a)(b)} [\![Q']\!]^\pi$. The premises of rule (close1) are all satisfied and we can use this rule to prove a transition $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{\tau} \nu([b]([\![P']\!]^\pi \parallel [\![Q']\!]^\pi))$, whose target is exactly $[\![\nu b.(P' \parallel Q')]\!]^\pi$.

- $!P$: Assume that $!P \xrightarrow{\alpha} P' \parallel !P$, with $P \xrightarrow{\alpha} P'$. Now, $[\![!P]\!]^\pi = !\,[\![P]\!]^\pi$. Since $P \xrightarrow{\alpha} P'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$. Thus the premise in rule (repl) is satisfied, and we can use this rule to prove a transition $!\,[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi \parallel !\,[\![P]\!]^\pi$, whose target is exactly $[\![P' \parallel !P]\!]^\pi$.

$\Leftarrow$

The proof is by induction on the structure of the π-term P and then by cases on the last rule used in the derivation of the transition $[\![P]\!]^\pi \xrightarrow{\alpha} Q$.

- $\mathbf{0}$: $[\![\mathbf{0}]\!]^\pi = \mathbf{0}$. Since α does not range over substitution and α-conversion labels, this case is vacuous.
- $\tau.P$: $[\![\tau.P]\!]^\pi = \tau.[\![P]\!]^\pi$. The only transition is $\tau.[\![P]\!]^\pi \xrightarrow{\tau} [\![P]\!]^\pi$ by rule (τ). In this case, $\tau.P \xrightarrow{\tau} P$ with $[\![\tau]\!]^\pi = \tau$, and we are done.
- $\overline{a}b.P$: $[\![\overline{a}b.P]\!]^\pi = out(a, b, [\![P]\!]^\pi)$. The only transition is $out(a, b, [\![P]\!]^\pi) \xrightarrow{out(a,b)} [\![P]\!]^\pi$, by rule (out). In this case, $\overline{a}b.P \xrightarrow{\overline{a}b} P$ with $[\![\overline{a}b]\!]^\pi = out(a, b)$ and we are done.
- $a(b).P$: $[\![a(b).P]\!]^\pi = in(a, [b][\![P]\!]^\pi)$. The only possible transitions have labels of the form $in(a, c)$. Let us pick an action $in(a, c)$. Then $in(a, [b][\![P]\!]^\pi) \xrightarrow{in(a,c)} P'$, by rule (in), with $[\![P]\!]^\pi \xrightarrow{a \mapsto c} P'$. By Lemma 5.16.1, which states the correctness of substitution transitions, $P' = [\![P[c/a]]\!]^\pi$. Since $[\![ac]\!]^\pi = in(a, c)$ and $\overline{a}b.P \xrightarrow{ac} P[c/a]$ we are done.
- $\nu b.P$: $[\![\nu a.P]\!]^\pi = \nu([a][\![P]\!]^\pi)$. There are two possible transitions.

- $\nu([b][\![P]\!]^\pi) \stackrel{bout(a,[b])}{\rightarrow} P'$ by rule (open), with $[\![P]\!]^\pi \stackrel{out(a,b)}{\rightarrow} P'$ and $a \neq b$. By the inductive hypothesis $[\![P']\!]^\pi = P''$ and $P \stackrel{\overline{a}b}{\rightarrow} P''$ for some P''. We can now use the rule (open) in order to prove a transition $\nu b.P \stackrel{\overline{a}(b)}{\rightarrow} P''$ with $[\![\overline{a}(b)]\!]^\pi = bout(a,[b])$ and we are done.
- $\nu([b][\![P]\!]^\pi) \stackrel{\alpha}{\rightarrow} \nu([b]P')$by rule (res), with $[\![P]\!]^\pi \stackrel{\alpha}{\rightarrow} P'$ and $b \notin bn(\alpha)$ and $b\#\alpha$. By inductive hypothesis $P \stackrel{\alpha'}{\rightarrow} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. By Lemma 5.16.4, since $b \notin bn(\alpha)$ and $b\#\alpha$ then we have that $b \notin names(\alpha')$. We can thus use the rule (resO) in order to prove a transition $\nu b.P \stackrel{\alpha'}{\rightarrow} \nu b.P''$ with $\alpha = [\![\alpha']\!]^\pi$ and we are done.

- $P + Q$: $[\![P + Q]\!]^\pi = [\![P]\!]^\pi + [\![Q]\!]^\pi$. There are two possible transitions, (1) $[\![P]\!]^\pi + [\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} P'$, by rule (sum1), with $[\![P]\!]^\pi \stackrel{\alpha}{\rightarrow} P'$, or (2) $[\![P]\!]^\pi + [\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} Q'$, by rule (sum2), with $[\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} Q'$. Here we consider only transition (1). By inductive hypothesis $P \stackrel{\alpha'}{\rightarrow} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $P + Q \stackrel{\alpha'}{\rightarrow} P''$, with $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. Case (2) can be proven analogously.

- $P \parallel Q$: $[\![P \parallel Q]\!]^\pi = [\![P]\!]^\pi \parallel [\![Q]\!]^\pi$. We distinguishes four cases:
  - $\alpha \notin \{\overline{a}(b) \mid a, b \in C\}$: There are two possible transitions, (1) $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} P' \parallel Q$, by rule (par1), with $[\![P]\!]^\pi \stackrel{\alpha}{\rightarrow} P'$, or (2) $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} P \parallel Q'$, by rule (par2), with $[\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} Q'$. (We treat below separately the case for $\alpha = \tau$ with a top level communication taking place). Here we consider only transition (1). By inductive hypothesis $P \stackrel{\alpha'}{\rightarrow} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $P \parallel Q \stackrel{\alpha'}{\rightarrow} P'' \parallel Q$, whose target is exactly $[\![P' \parallel Q]\!]^\pi$ and with $\alpha = [\![\alpha']\!]^\pi$. Case (2) can be proven analogously.
  - $\alpha = \overline{a}(b)$: There are two possible transitions: (1) $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \stackrel{\alpha}{\rightarrow} P' \parallel Q$, by rule (parRes1), with $[\![P]\!]^\pi \stackrel{bout(a,[b])}{\rightarrow} P'$ and b fresh in $[\![Q]\!]^\pi$, or (2) $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \stackrel{bout(a,[b])}{\rightarrow} P \parallel Q'$, by rule (parRes2), with $[\![Q]\!]^\pi \stackrel{bout(a,[b])}{\rightarrow} Q'$ and b fresh in $[\![P]\!]^\pi$, Here we consider only transition (1). By the inductive hypothesis, $P' = [\![P'']\!]^\pi$ and $P \stackrel{\overline{a}(b)}{\rightarrow} P''$ for some P''. By Lemma 5.16.5, since b is fresh in $[\![Q]\!]^\pi$ we have that $b \notin fn(Q)$. We can thus prove a transition $P \parallel Q \stackrel{\overline{a}(b)}{\rightarrow} P'' \parallel Q$, whose target is exactly $[\![P' \parallel Q]\!]^\pi$, and with $[\![\overline{a}(b)]\!]^\pi = bout(a,[b])$.
  - $\alpha = \tau$ using (com1): $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \stackrel{\tau}{\rightarrow} P' \parallel Q'$, with $[\![P]\!]^\pi \stackrel{out(a,b)}{\rightarrow} P'$ and $[\![Q]\!]^\pi \stackrel{in(a,b)}{\rightarrow} Q'$. By the inductive hypothesis, $P' = [\![P'']\!]^\pi$and $P \stackrel{\overline{a}b}{\rightarrow} P''$ for

some P'', and also $[\![Q']\!]^\pi = Q''$ and $Q \xrightarrow{ab} Q''$ for some Q''. We can thus use the rule (comO1) in order to prove a transition $P \parallel Q \xrightarrow{\tau} P'' \parallel Q''$, whose target is exactly $[\![P' \parallel Q']\!]^\pi$, and with $[\![\tau]\!]^\pi = \tau$. When using the rule (com2) the same reasoning applies.

- $\alpha = \tau$ using (close1): $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{\tau} \nu([b](P' \parallel Q'))$, with $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} P'$, $[\![Q]\!]^\pi \xrightarrow{in(a,b)} Q'$ and b fresh in both $[\![P]\!]^\pi$ and $[\![Q]\!]^\pi$. By the inductive hypothesis, $P' = [\![P'']\!]^\pi$ and $P \xrightarrow{\bar{a}(b)} P''$ for some P'', and also $[\![Q']\!]^\pi = Q''$ and $Q \xrightarrow{ab} Q''$ for some Q''. By Lemma 5.16.5, since b is fresh in $[\![Q]\!]^\pi$ we have that $b \notin fn(Q)$. We can thus use the rule (closeO1) in order to prove a transition $P \parallel Q \xrightarrow{\tau} \nu b.(P'' \parallel Q'')$, whose target is exactly $[\![\nu([b](P' \parallel Q'))]\!]^\pi$, and with $[\![\tau]\!]^\pi = \tau$. When using the rule (close2) the same reasoning applies.

- $!P$: $[\![!P]\!]^\pi = !\,[\![P]\!]^\pi$. The only transition is $!\,[\![P]\!]^\pi \xrightarrow{\alpha} P' \parallel !\,[\![P]\!]^\pi$, by rule (repl), with $[\![P]\!]^\pi \xrightarrow{\alpha} P'$. By inductive hypothesis $P \xrightarrow{\alpha'} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $!P \xrightarrow{\alpha'} P'' \parallel !P$, whose target is exactly $[\![P' \parallel !P]\!]^\pi$, and with $\alpha = [\![\alpha']\!]^\pi$.

5.17 Correctness of substitutions for λ: Proof of Lemma 5.15.1

We prove the two implications in the statement of Lemma 5.15.1 separately.

- $\Rightarrow$: $[\![M]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M[N/a]]\!]^\lambda$
- $\Leftarrow$: if $M[N/a] = M'$ then $[\![M]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M']\!]^\lambda$

$\Rightarrow$

Let us pick closed λ-terms M and N and an atom a. The proof is by induction on the structure of the λ-term M.

- a: We have that $[\![a]\!]^\lambda = a$. In this case $a[N/a] = N$. We can use rule ($a1_{Ts}$) to prove $a \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![N]\!]^\lambda$.
- b, with $b \neq a$: In this case $b[N/a] = b$. We can use rule ($a1_{Ts}$) to prove $b \xrightarrow{a \mapsto [\![N]\!]^\lambda} b$.

- $\lambda a.M$: We have that $[\![\lambda a.M]\!]^\lambda = \lambda([a][\![M]\!]^\lambda)$. In this case $(\lambda a.M)[N/a] = \lambda a.M$. We can use rule $(\text{abs2}_{\text{Ts}})$ to prove $\lambda([a][\![M]\!]^\lambda) \xrightarrow{a \mapsto [\![N]\!]^\lambda} \lambda([a][\![M]\!]^\lambda)$, whose target is exactly $[\![\lambda a.M]\!]^\lambda$.
- $\lambda b.M$, with $b \neq a$ and b is not fresh in N: In this case the substitution $(\lambda b.M)[N/a]$ is not possible in order to avoid capture. In our formulation of the λ-calculus a transition $\lambda([a][\![M]\!]^\lambda) \xrightarrow{b \mapsto [\![N]\!]^\lambda} M'$ is not provable, for any closed term M'. Indeed, the rules (abs1_{s}) can not be applied because of the freshness test and rules (abs2_{s}) apply only when $b = a$, that is not the case. This case is therefore vacuous.
- $\lambda b.M$, with $b \neq a$ and b fresh in N: In this case $(\lambda b.M)[N/a] = \lambda b.(M[N/a])$. Now, by the inductive hypothesis, which applies to M, we know that $[\![M]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M[N/a]]\!]^\lambda$. We thus can use the rule $(\text{abs1}_{\text{Ts}})$ to prove $\lambda([b][\![M]\!]^\lambda) \xrightarrow{b \mapsto [\![N]\!]^\lambda} \lambda([b][\![M[N/a]]\!]^\lambda)$, whose target is exactly $[\![\lambda b.M[N/a]]\!]^\lambda$.
- $(M_1\ M_2)$: We have that $[\![(M_1\ M_2)]\!]^\lambda = ([\![M_1]\!]^\lambda\ [\![M_2]\!]^\lambda)$ and $(M_1\ M_2)[N/a] = (M_1[N/a]\ M_2[N/a])$. Now, by the inductive hypothesis which applies to M_1 and M_2, we know that $[\![M_1]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M_1[N/a]]\!]^\lambda$ and $[\![M_2]\!]^\lambda \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M_2[N/a]]\!]^\lambda$. We thus can use the rule (app_{s}) to prove the transition

  $$([\![M_1]\!]^\lambda\ [\![M_2]\!]^\lambda) \xrightarrow{a \mapsto [\![N]\!]^\lambda} [\![M_1[N/a]]\!]^\lambda\ [\![M_2[N/a]]\!]^\lambda,$$

  whose target is exactly $[\![(M_1[N/a]\ M_2[N/a])]\!]^\lambda$.

$\Leftarrow$

The case $\Leftarrow$ can be proved following the lines of the proof above.

5.18 Correctness of substitutions for π: Proof of Lemma 5.16.1

The proof of Lemma 5.16.1 is divided into two cases.

- $\Rightarrow$: $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P[b/a]]\!]^\pi$
- $\Leftarrow$: if $P[b/a] = P'$ then $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P']\!]^\pi$

$\Rightarrow$

Let us pick a closed π-terms P and atoms a and b. The proof is by induction on the structure of the π-term P.

- $\mathbf{0}$: We have that $[\![\mathbf{0}]\!]^\pi = \mathbf{0}$ and $\mathbf{0}[b/a] = \mathbf{0}$. The reader should bear in mind that, concerning $\mathbf{0}$, since it is a function symbol with arity 0, the instanciation of the rule scheme ($\mathrm{f_s}$) of Section 5.4.1 for $\mathbf{0}$ is a set of rules with no premises, i.e. an axiom $\mathbf{0} \xrightarrow{a \mapsto b} \mathbf{0}$ for all atoms a and b. We can thus prove the transition $\mathbf{0} \xrightarrow{a \mapsto b} \mathbf{0}$, which is exactly $[\![\mathbf{0}]\!]^\pi$.

- $\tau.P$: We have that $[\![\tau.P]\!]^\pi = \tau.[\![P]\!]^\pi$ and $(\tau.P)[b/a] = \tau.P[b/a]$. By inductive hypothesis, which applies to P, we know that $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P[b/a]]\!]^\pi$. We can thus use the rule (τ_s) in order to prove the transition $\tau.[\![P]\!]^\pi \xrightarrow{a \mapsto b} \tau.[\![P[b/a]]\!]^\pi$, which is exactly $[\![\tau.P[b/a]]\!]^\pi$.

- $\overline{c}d.P$, with a, b, c, d not necessarily pairwise distinct atoms: We have that $[\![\overline{c}d.P]\!]^\pi = out(c, d, [\![P]\!]^\pi)$ and $(\overline{c}d.P)[b/a] = \overline{c[b/a]}d[b/a].P[b/a]$. By inductive hypothesis, which applies to P, and to the atoms c and d, we know that $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P[b/a]]\!]^\pi$, $c \xrightarrow{a \mapsto b} [\![c[b/a]]\!]^\pi$ and $d \xrightarrow{a \mapsto b} [\![d[b/a]]\!]^\pi$. We can thus use the rule ($\mathrm{out_s}$) in order to prove the transition $out(c, d, [\![P]\!]^\pi) \xrightarrow{a \mapsto b} out([\![c[b/a]]\!]^\pi, [\![d[b/a]]\!]^\pi, [\![P[b/a]]\!]^\pi)$, which is exactly $[\![\overline{c[b/a]}d[b/a].P[b/a]]\!]^\pi$.

- $c(d).P$, with $a, b \neq d$: We have that $[\![\overline{c}d.P]\!]^\pi = out(c, d, [\![P]\!]^\pi)$ and $c(d).P[b/a] = c[b/a](d).P[b/a]$. By inductive hypothesis, which applies to P, and to the atom c, we know that $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P[b/a]]\!]^\pi$ and $c \xrightarrow{a \mapsto b} [\![c[b/a]]\!]^\pi$. We can thus use the rule ($abs1_{\mathrm{Ts}}$) to prove the transition $[d][\![P]\!]^\pi \xrightarrow{a \mapsto b} [d][\![P[b/a]]\!]^\pi$. Being this transition provable, we can use the rule ($\mathrm{in_s}$) in order to prove the transitions $in(c, [d][\![P]\!]^\pi) \xrightarrow{a \mapsto b} in([\![c[b/a]]\!]^\pi, [d][\![P[b/a]]\!]^\pi)$, which is exactly $[\![c[b/a](d).P[b/a]]\!]^\pi$.

- $c(a).P$, with with $b \neq d$: We have that $[\![\overline{c}a.P]\!]^\pi = out(c, a, [\![P]\!]^\pi)$ and $c(a).P[b/a]$ is not possible in order to avoid capture. In our formulation of π a transition $in(c, [a][\![P]\!]^\pi) \xrightarrow{a \mapsto b} P'$, is not provable, for any closed term P'. Indeed, the rules ($\mathrm{abs1_s}$) generated in Section 5.4.1 specifically apply only for atoms different from a, and rules ($\mathrm{abs2_s}$) only when $b = a$, that is not the case. This case is therefore vacuous.

- $c(b).P$: We have that $[\![\overline{c}b.P]\!]^\pi = out(c, b, [\![P]\!]^\pi)$ and $c(b).P[b/a] = c[b/a](b).P$. We can use the rule ($abs2_{\mathrm{Ts}}$) in order to prove the transition $[d][\![P]\!]^\pi \xrightarrow{a \mapsto b} [b][\![P]\!]^\pi$. Being this transition provable, and knowing that by inductive hypothesis

$c \xrightarrow{a\mapsto b} [\![c[b/a]]\!]^\pi$, we can use the rule ($\text{in}_\text{s}$) in order to prove the transition $in(c, [b][\![P]\!]^\pi) \xrightarrow{a\mapsto b} in([\![c[b/a]]\!]^\pi, [b][\![P]\!]^\pi)$, which is exactly $[\![c(b).P[b/a]]\!]^\pi$.

- $\nu a.P$: We have that $[\![\nu a.P]\!]^\pi = \nu([a][\![P]\!]^\pi)$ and $(\nu a.P)[b/a] = \nu a.P$. We can use the rule (abs2_{Ts}) in order to prove the transition $[a][\![P]\!]^\pi \xrightarrow{a\mapsto b} [a][\![P]\!]^\pi$. Being this transition provable, we can use the rule (in_s) in order to prove the transition $\nu([a][\![P]\!]^\pi) \xrightarrow{a\mapsto b} \nu([a][\![P]\!]^\pi)$, which is exactly $[\![\nu a.P]\!]^\pi$.

- $\nu b.P$, with $b \neq a$. In this case the substitution $(\nu b.P)[b/a]$ is not possible in order to avoid capture. In our formulation of the π-calculus a transition $[b][\![P]\!]^\pi \xrightarrow{a\mapsto b} P'$ is not provable, for any closed term P'. Indeed, the rules (abs1_s) can not be applied because of the freshness test and rules (abs2_s) apply only when $b = a$, that is not the case. This case is therefore vacuous.

- $\nu c.P$, with $c \neq a$ and $c \neq b$: We have that $[\![\nu c.P]\!]^\pi = \nu([c][\![P]\!]^\pi)$ and $(\nu c.P)[b/a] = \nu c.P[b/a]$. By inductive hypothesis, which applies to P, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$. We use the rule ($abs1_{\text{Ts}}$) to prove the transition $[c][\![P]\!]^\pi \xrightarrow{a\mapsto b} [c][\![P[b/a]]\!]^\pi$. Using this transition, we use the rule (ν_s) to prove the transition $\nu([c][\![P]\!]^\pi) \xrightarrow{a\mapsto b} \nu([c][\![P[b/a]]\!]^\pi)$, which is exactly $[\![\nu c.P[b/a]]\!]^\pi$.

- $P + Q$: We have that $[\![P + Q]\!]^\pi = [\![P]\!]^\pi + [\![Q]\!]^\pi$ and $P + Q[b/a] = P[b/a] + Q[b/a]$. By inductive hypothesis, which applies to P and Q, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![Q[b/a]]\!]^\pi$. We can thus use the rule ($+_\text{s}$) to prove the transition $[\![P]\!]^\pi + [\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi + [\![Q[b/a]]\!]^\pi$, which is exactly $[\![P[b/a] + Q[b/a]]\!]^\pi$.

- $P \parallel Q$: We have that $[\![P \parallel Q]\!]^\pi = [\![P]\!]^\pi \parallel [\![Q]\!]^\pi$ and $P \parallel Q[b/a] = P[b/a] \parallel Q[b/a]$. By inductive hypothesis, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![Q[b/a]]\!]^\pi$. We use the rule ($\parallel_\text{s}$) to prove the transition $[\![P]\!]^\pi \parallel [\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi \parallel [\![Q[b/a]]\!]^\pi$, which is exactly $[\![P[b/a] \parallel Q[b/a]]\!]^\pi$.

- !P: We have that $[\![!P]\!]^\pi = !\![\![P]\!]^\pi$ and $(!P)[b/a] = !(P[b/a])$. By inductive hypothesis, which applies to P, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$. We can thus use the rule ($!_\text{s}$) to prove the transition $![\![P]\!]^\pi \xrightarrow{a\mapsto b} ![\![P[b/a]]\!]^\pi$, which is exactly $[\![!(P[b/a])]\!]^\pi$.

$\Leftarrow$

The case $\Leftarrow$ can be proved following the lines of the proof above.

5.19 Correctness of α-conversions for λ: Proof of Lemma 5.15.2

The proof of Lemma 5.15.2 is divided into two cases.

- $\Rightarrow$: if $[\![M]\!]^\lambda \rightarrow_\alpha N'$ then $M \equiv N \wedge [\![N]\!]^\lambda = N'$.
- $\Leftarrow$: if $M \equiv N$ then $[\![M]\!]^\lambda \rightarrow_\alpha [\![N]\!]^\lambda$.

Where $\equiv$ is the syntactic equality up to α-equivalence of the lazy λ-calculus.

$\Rightarrow$

Assuming the hypothesis, let T be a $NTSS$. The proof is by induction on the length of the proofs for α-conversion transitions.

- Proofs of length 1: The only provable transition with length 1 is by rule (id_α), $[\![M]\!]^\lambda \rightarrow_\alpha [\![M]\!]^\lambda$. Indeed, by reflexivity of $\equiv$, we have that $M \equiv M$.
- Proofs of length n:
  - $[\![\lambda a.M]\!]^\lambda$: There are two possible provable α-conversion transitions for $[\![\lambda a.M]\!]^\lambda$.
    - $[\![\lambda a.M]\!]^\lambda \rightarrow_\alpha \lambda([b]M')$, with b fresh in $[\![M]\!]^\lambda$: Here we first prove $[a][\![M]\!]^\lambda \rightarrow_\alpha [b]M'$ using rule ($\mathrm{abs1}_\alpha$) and then $\lambda([a][\![M]\!]^\lambda) \rightarrow_\alpha \lambda([b]M')$ using the rule (λ_α). Rule ($\mathrm{abs1}_\alpha$) can be instanciated in the following way

$$\frac{[\![M]\!]^\lambda \xrightarrow{a \mapsto b} M' \qquad b \# [\![M]\!]^\lambda}{[a][\![M]\!]^\lambda \rightarrow_\alpha [b]M'}$$

      By Theorem 5.17, which states the correctness of substitution transitions, we know that $M' = [\![M[b/a]]\!]^\lambda$. The transition actually proved by the rule above is thus $[a][\![M]\!]^\lambda \rightarrow_\alpha [b][\![M[b/a]]\!]^\lambda$. Now, we can instanciate the rule (λ_α) as follows

$$\frac{[a][\![M]\!]^\lambda \rightarrow_\alpha [b][\![M[b/a]]\!]^\lambda}{\lambda([a][\![M]\!]^\lambda) \rightarrow_\alpha \lambda([b][\![M[b/a]]\!]^\lambda)}$$

in order to prove $\lambda([a]\llbracket M\rrbracket^\lambda) \to_\alpha \lambda([b]\llbracket M[b/a]\rrbracket^\lambda)$. The statement of the theorem holds, indeed $\lambda a.M \equiv \lambda b.M[b/a]$.

* $\llbracket \lambda a.M\rrbracket^\lambda \to_\alpha \lambda([a]M')$: Here we first prove $[a]\llbracket M\rrbracket^\lambda \to_\alpha [a]M'$ using rule (abs2$_\alpha$) and then $\lambda([a]\llbracket M\rrbracket^\lambda) \to_\alpha \lambda([a]M')$ using the rule λ_α. Rule (abs2$_\alpha$) can be instanciated in the following way

$$\frac{\llbracket M\rrbracket^\lambda \to_\alpha M'}{[a]\llbracket M\rrbracket^\lambda \to_\alpha [a]M'}$$

Since the proof length for proving this transition is strictly less than n, also the proof length for proving $\llbracket M\rrbracket^\lambda \to_\alpha M'$ is strictly less than n. By inductive hypothesis we can conclude that $M \equiv M''$, with $\llbracket M''\rrbracket^\lambda = M'$. Now, we can instanciate the rule λ_α as follows

$$\frac{[a]\llbracket M\rrbracket^\lambda \to_\alpha [a]\llbracket M''\rrbracket^\lambda}{\lambda([a]\llbracket M\rrbracket^\lambda) \to_\alpha \lambda([a]\llbracket M''\rrbracket^\lambda)}$$

in order to prove $\lambda([a]\llbracket M\rrbracket^\lambda) \to_\alpha \lambda([a]\llbracket M''\rrbracket^\lambda)$. The statement of the theorem holds in this case. Indeed, since $\equiv$ is a congruence, we can place equated terms in the same context, and since $M \equiv M''$ we have that $\lambda a.M \equiv \lambda a.M''$.

– $(\llbracket M_1\rrbracket^\lambda\ \llbracket M_2\rrbracket^\lambda) \to_\alpha (M_1'\ M_2')$ using rule (app$_\alpha$), with $\llbracket M_1\rrbracket^\lambda \to_\alpha M_1'$ and $\llbracket M_2\rrbracket^\lambda \to_\alpha M_2'$: Since the proof length for proving these two mentioned transitions is strictly less than n, by inductive hypothesis we can conclude that $M_1 \equiv M_1''$ and $M_2 \equiv M_2''$, with $\llbracket M_1''\rrbracket^\lambda = M_1'$ and $\llbracket M_2''\rrbracket^\lambda = M_2'$. As in the previous case, since $\equiv$ is a congruence, $(M_1\ M_2) \equiv (M_1''\ M_2'')$, with exactly $\llbracket (M_1''\ M_2'')\rrbracket^\lambda = (M_1'\ M_2')$.

– $\llbracket M_1\rrbracket^\lambda \to_\alpha M_3'$ using rule $\alpha \cdot \text{upTo}\alpha$, with $\llbracket M_1\rrbracket^\lambda \to_\alpha M_2'$ and $M_2' \to_\alpha M_3'$: Since the proof length for proving the transition $\llbracket M_1\rrbracket^\lambda \to_\alpha M_2'$ is strictly less than n, by inductive hypothesis we can conclude that $M_1 \equiv M_2$, with $\llbracket M_2\rrbracket^\lambda = M_2'$. Again, the proof length for proving the transition $M_2' \to_\alpha M_3'$, which is actually $\llbracket M_2\rrbracket^\lambda \to_\alpha M_3'$, is strictly less than n, and by inductive hypothesis we can conclude that $M_2 \equiv M_3$, with $\llbracket M_3\rrbracket^\lambda = M_3'$. Now, having $M_1 \equiv M_2$ and $M_2 \equiv M_3$, since $\equiv$ is a congruence, it is a transitive relation, and thus we can conclude $M_1 \equiv M_3$.

$\Leftarrow$

The case $\Leftarrow$ is proved along the line of the case $\Leftarrow$ in proof of Theorem 5.4.4. Basically, the change of the bound variable in binders is simulated by rule (abs1_α), the reflexivity is given by rule (id_α), the symmetry is inferred, and the transitivity is given by the rule $\alpha \cdot \text{upTo}\alpha$. Moreover, the reader can see that the rules are such to share α-conversion transitions in any context, which simulates the requirement for $\equiv$ to be a congruence and not just an equivalence relation.

5.20 Correctness of α-conversions for π: Proof of Lemma 5.16.2

The proof of Lemma 5.16.2 is divided into two cases.

- $\Rightarrow$: if $[\![P]\!]^\pi \rightarrow_\alpha Q'$ then $P \equiv Q \wedge [\![Q]\!]^\pi = Q'$.
- $\Leftarrow$: if $P \equiv Q$ then $[\![P]\!]^\pi \rightarrow_\alpha [\![Q]\!]^\pi$.

Where $\equiv$ is the syntactic equality up to α-equivalence of the early π-calculus.

$\Rightarrow$

Assuming the hypothesis, let T be a $NTSS$. The proof is by induction on the length of the proofs for α-conversion transitions.

- Proofs of length 1: The only provable transition with length 1 is by rule (id_α), $[\![P]\!]^\pi \rightarrow_\alpha [\![P]\!]^\pi$. Indeed, by reflexivity of $\equiv$, we have that $P \equiv P$.
- Proofs of length n: The rest of the proof proceed in as much the same way as in Lemma 5.15.2. In what follows we thus show the proof only for one binder of the ordinary π, namely the restriction $\nu a.P$, and one ordinary operator, namely the parallel operator $\|$. The proofs regarding the other binder $a(b).P$, the other operators and the transitivity case are easy to carry out following the same line employed in detail in the proof of Lemma 5.15.2.
  - $[\![\nu a.P]\!]^\pi$: There are two possible provable α-conversion transitions for $[\![\nu a.P]\!]^\pi$.
    - $[\![\nu a.P]\!]^\pi \rightarrow_\alpha \nu([b]P')$, with b fresh in $[\![P]\!]^\pi$: Here we first prove $[a][\![P]\!]^\pi \rightarrow_\alpha [b]P'$ using rule (abs1_α) and then $\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([b]P')$

using the rule (ν_α). Rule (abs1_α) can be instanciated in the following way

$$\frac{[\![P]\!]^\pi \xrightarrow{a\mapsto b} P' \qquad b\#[\![P]\!]^\pi}{[a][\![P]\!]^\pi \rightarrow_\alpha [b]P'}$$

By Theorem 5.18, which states the correctness of substitution transitions, we know that $P' = [\![P[b/a]]\!]^\pi$. The transition actually proved by the rule above is thus $[a][\![P]\!]^\pi \rightarrow_\alpha [b][\![P[b/a]]\!]^\pi$. Now, we can instanciate the rule (ν_α) as follows

$$\frac{[a][\![P]\!]^\pi \rightarrow_\alpha [b][\![P[b/a]]\!]^\pi}{\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([b][\![P[b/a]]\!]^\pi)}$$

in order to prove $\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([b][\![P[b/a]]\!]^\pi)$. The statement of the theorem holds in this case. Indeed, $\nu a.P \equiv \nu b.P[b/a]$.

* $[\![\nu a.P]\!]^\pi \rightarrow_\alpha \nu([a]P')$: Here we address the case where we first prove $[a][\![P]\!]^\pi \rightarrow_\alpha [a]P'$ using rule (abs2_α) and then $\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([a]P')$ using the rule (ν_α). Rule (abs2_α) can be instanciated in the following way

$$\frac{[\![P]\!]^\pi \rightarrow_\alpha P'}{[a][\![P]\!]^\pi \rightarrow_\alpha [a]P'}$$

Since the proof length for proving this transition is strictly less then n, also the proof length for proving $[\![P]\!]^\pi \rightarrow_\alpha P'$ is strictly less then n. By inductive hypothesis we can conclude that $P \equiv P''$, with $[\![P'']\!]^\pi = P'$. Now, we can instanciate the rule (ν_α) as follows

$$\frac{[a][\![P]\!]^\pi \rightarrow_\alpha [a][\![P'']\!]^\pi}{\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([a][\![P'']\!]^\pi}$$

in order to prove $\nu([a][\![P]\!]^\pi) \rightarrow_\alpha \nu([a][\![P'']\!]^\pi$. The statement of the theorem holds in this case. Indeed, since $\equiv$ is a congruence, we can place equated terms in the same context, and since $P \equiv P''$ we have that $\nu a.P \equiv \nu a.P''$.

– $[\![P_1]\!]^\pi \parallel [\![P_2]\!]^\pi \rightarrow_\alpha P_1' \parallel P_2'$ using rule ($\parallel_\alpha$), with $[\![P_1]\!]^\pi \rightarrow_\alpha P_1'$ and $[\![P_2]\!]^\pi \rightarrow_\alpha P_2'$: Since the proof length for proving these two mentioned transitions is strictly less than n, by inductive hypothesis we can conclude that $P_1 \equiv P_1''$ and $P_2 \equiv P_2''$, with $[\![P_1'']\!]^\pi = P_1'$ and $[\![P_2'']\!]^\pi = P_2'$. As in the previous case, since $\equiv$ is a congruence, $P_1 \parallel P_2 \equiv P_1'' \parallel P_2''$, with exactly $[\![P_1'' \parallel P_2'']\!]^\pi = P_1' \parallel P_2'$.

$\Leftarrow$

The case $\Leftarrow$ is proved along the line of the case $\Leftarrow$ in proof of Theorem 5.4.4. Basically, the change of the bound variable in binders is simulated by rule (abs1$_\alpha$), the reflexivity is given by rule (id$_\alpha$), the symmetry is inferred, and the transitivity is given by the rule $\alpha \cdot \text{upTo}\alpha$. Moreover, the reader can see that the rules are such to share α-conversion transitions in any context, which simulates the requirement for $\equiv$ to be a congruence and not just an equivalence relation.

5.21 Bisimilarity when ignoring substitution transitions: Proof of Theorem 5.6.3

The proof is divided into two cases, given P and $Q \in \Pi$:

1. Soundness: if $P \leftrightarrows Q$ then $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$.
2. Completeness: if $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$ then $P \leftrightarrows Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{([\![P]\!]^\pi, [\![Q]\!]^\pi) \mid P \leftrightarrows Q\}$$

satisfies the requirement for $\underline{\leftrightarrow}^-$, i.e., the ones in the definition of the nominal bisimilarity (Definition 5.6.2) when we omit to consider substitution transitions.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $[\![P]\!]^\pi \mathcal{R} [\![Q]\!]^\pi$ and $[\![P]\!]^\pi \xrightarrow{\alpha} P'$. In our formulation of the early π-calculus, the action α' may be either (1) an ordinary action, (2) a substitution transition, or (3) an α-conversion

transition. Since $\underline{\leftrightarrow}^-$ omits to match substitution transitions, we only need to tackle transitions of form (1) and (3).

Let us consider now the case (1). By Theorem 5.5.2, which states the operational correctness of our formulation of π-calculus with respect to the original one, we know that $P \xrightarrow{\alpha'} P''$ for some P'' and α such that $[\![P'']\!]^\pi = P'$ and $[\![\alpha']\!]^\pi = \alpha$. Since $P \leftrightarrows Q$, we have that $Q \xrightarrow{\alpha'} Q''$, with $P'' \leftrightarrows Q''$. By, again, Theorem 5.5.2, we have that $[\![Q]\!]^\pi \xrightarrow{[\![\alpha']\!]^\pi} [\![Q'']\!]^\pi$, i.e. $[\![Q]\!]^\pi \xrightarrow{\alpha'} [\![Q'']\!]^\pi$. We have now to prove that $[\![P'']\!]^\pi \mathcal{R} [\![Q'']\!]^\pi$. This follows easily from the fact that $P'' \leftrightarrows Q''$.

Let us now consider the case (3), namely the transition is an α-conversion transition. Since $[\![P]\!]^\pi \rightarrow_\alpha P'$, by Lemma 5.16.2, which states the correctness of α-conversion transitions, we know that $P \equiv P''$ and $[\![P'']\!]^\pi = P'$. We now have to show that $[\![Q]\!]^\pi \rightarrow_\alpha Q'$ for some Q' such that $P' \mathcal{R} Q'$. Since $\equiv \subset \leftrightarrows$, we have that there exists a Q'' such that $Q \equiv Q''$ and it holds that $P'' \leftrightarrows Q''$. Since $Q \equiv Q''$, by Lemma 5.16.2 we have that $[\![Q]\!]^\pi \rightarrow_\alpha Q'$, with $[\![Q'']\!]^\pi = Q'$. We have now to prove that $[\![P'']\!]^\pi \mathcal{R} [\![Q'']\!]^\pi$. This follows easily from the fact that $P'' \leftrightarrows Q''$.

Completeness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid [\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi\}$$

is a bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$. By Theorem 5.5.2, which states the operational correctness of our formulation of π-calculus with respect to the original one, we know that $[\![P]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} [\![P']\!]^\pi$. Since $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$, we also have that $[\![Q]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} Q''$, for some Q'' such that $[\![P']\!]^\pi \underline{\leftrightarrow}^- Q''$, and by Theorem 5.5.2 we know that Q'' is such that $Q \xrightarrow{\alpha} Q'$ with $[\![Q']\!]^\pi = Q''$. We have now to prove that $P' \mathcal{R} Q'$. This follows easily from the fact that $[\![P']\!]^\pi \underline{\leftrightarrow}^- [\![Q']\!]^\pi$.

5.22 Open bisimilarity and Bisimilarity coincide: Proof of Theorem 5.6.6

In this section we prove that what open bisimilarity does in the ordinary early π-calculus is exactly what nominal bisimilarity does in our formulation of the early π-calculus in the nominal SOS framework.

In order to prove this, we first define an equivalence relation over terms of the ordinary π-calculus, we call it *One-Step Open Bisimilarity*. This equivalence requires the ordinary bisimilarity, as recalled in Definition 5.6.1, to match also all the "single substitutions" performed from the processes to be equated. We prove that the nominal bisimilarity in our formulation of π-calculus coincides with the One-Step Open Bisimilarity. We then prove that the One-Step Open Bisimilarity is another way to formulate the open bisimilarity of Definition 5.6.5. The statement of Theorem 5.6.6 then easily follows.

Definition 5.22.1 (One-Step Open Bisimilarity) One-Step open bisimilarity $\underline{\leftrightarrow}^{1sO}$ *is the largest symmetric relation* $\sim$ *between* π*-calculus processes such that whenever* $P \sim Q$,

1. *for all actions* α, *if* $P \xrightarrow{\alpha} P'$, *then there exists some* Q', *such that* $Q \xrightarrow{\alpha} Q'$ *and* $P' \sim Q'$;
2. *for all channel names* a *and* b, $P[b/a] \sim Q[b/a]$.

Theorem 5.22.2 (One-Step Open bisimilarity and Bisimilarity coincide) *For all* $P, Q \in \Pi$, $P \underline{\leftrightarrow}^{1sO} Q$ *if, and only if,* $[\![P]\!]^{\pi} \leftrightarrows [\![Q]\!]^{\pi}$.

The proof of this theorem is divided into two cases:

1. Soundness: if $P \underline{\leftrightarrow}^{1sO} Q$ then $[\![P]\!]^{\pi} \leftrightarrows [\![Q]\!]^{\pi}$.
2. Completeness: if $[\![P]\!]^{\pi} \leftrightarrows [\![Q]\!]^{\pi}$ then $P \underline{\leftrightarrow}^{1sO} Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{([\![P]\!]^{\pi}, [\![Q]\!]^{\pi}) \mid P \underline{\leftrightarrow}^{1sO} Q\}$$

is a nominal bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $[\![P]\!]^{\pi} \mathcal{R} [\![Q]\!]^{\pi}$ and $[\![P]\!]^{\pi} \xrightarrow{\alpha} P'$. In our formulation, the label α can perform be (1) an ordinary action,

(2) a substitution transition, or (3) an α-conversion transition. For cases (1) and (3) the exact reasoning employed for them in the proof of Theorem 5.6.3 applies. It thus suffices considering the case (2), which is about substitution transitions. In order to prove this, we rely on Lemma 5.16.1 which states the correctness of substitution transitions, and prove that for all names a and b, $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![Q[b/a]]\!]^\pi$, with $[\![P[b/a]]\!]^\pi \leftrightarrow [\![Q[b/a]]\!]^\pi$. The reader can easily notice that this follows immediately. Given atoms a and b, by Clause 2 of Definition 5.22.1 we know that $P[b/a] \underline{\leftrightarrow}^{1sO} Q[b/a]$, thus $[\![P[b/a]]\!]^\pi \mathcal{R} [\![Q[b/a]]\!]^\pi$.

Completeness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid [\![P]\!]^\pi \underline{\leftrightarrow}^{-} [\![Q]\!]^\pi\}$$

is a one-step open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P\mathcal{R}Q$.

Clause 1 of Definition 5.22.1 requires that P and Q would match transitions. For this case the exact reasoning employed in the proof of Theorem 5.6.3 applies.

Clause 2 of Definition 5.22.1 concerns substitutions. We have to prove that for all channel names a and b $P[b/a]\mathcal{R}Q[b/a]$. Let us pick two names a and b, by Lemma 5.16.1 which states the correctness of substitution transitions, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![Q[b/a]]\!]^\pi$. Since $[\![P]\!]^\pi \leftrightarrow [\![Q]\!]^\pi$ we moreover know that $[\![P[b/a]]\!]^\pi \leftrightarrow [\![Q[b/a]]\!]^\pi$. Thus, we can conclude that $P[b/a]\mathcal{R}Q[b/a]$.

Theorem 5.22.3 (Open and One-Step Open bisimilarity coincide) *For all $P, Q \in \Pi$, $P \mathrel{\hat{\leftrightarrow}} Q$ if, and only if, $P \underline{\leftrightarrow}^{1sO} Q$.*

Before embarking ourselves in the proof of Theorem 5.22.3, some preliminary considerations are in order. The completeness part of the proof of Theorem 5.22.3 relies on the fact that we can faithfully simulate the substitutions involved in the open bisimilarity, see Definition 5.6.4, by means of a sequence of substitutions that replace only one name with another. However, it is to be noticed that the substitutions of Definition 5.6.4 defines a mapping that replaces the names of a process simultaneously. For this reason, these substitutions will be called from now onwards *simultaneous substitutions*. The substitutions that replace only one atom with another within a process will be referred to as *one-step substitutions*.

The reader must see that an encoding from simultaneous to one-step substitutions cannot be provided naively. Consider for instance the term $a.b.c$[12] and a substitution σ which maps the name a to b[13], b to c, c to a and is the identity over all other atoms. The scenario we have is the following

$$(a.b.c)\sigma = b.c.a \qquad a.b.c[b/a][c/b][a/c] = a.a.a$$

The substitution σ indeed replaces names in the processes simultaneously. This fact cannot be simulated naively by one-step substitutions because of clashes of names in the terms. The unfolded procedure of the example above is $a.b.c[b/a] = b.b.c$, $b.b.c[c/b] = c.c.c$, $c.c.c[a/c] = a.a.a$. In the encoding of simultaneous substitutions that we present, we are able to simulate substitutions relying, not surprisingly, on freshness of atoms, which once again plays a crucial role.

In order to ease the proof and its presentation, we adopt a convenient representation of simultaneous substitutions. By Definition 5.6.4, we have that this type of substitutions act only on a finite set of names, we can thus represent mappings as finite lists of substitutions $\{a/b\}$, with a and b names. For instance, the substitution mapping a to b, c to d, e to f and that is the identity over all other names, is represented as $\{a/b\} \cdot \{c/d\} \cdot \{e/f\} \cdot \epsilon$. Such a representation gives a clearer and more immediate presentation. The symbol ϵ denotes the empty substitution for both the types of substitution. The operation $\cdot$ denotes the composition for simultaneous substitutions. For one-step substitution we write instead $t[a/b][c/d]$ as before. In what follows, we sometimes omit writing the substitution ϵ at the end of a composition of simultaneous substitutions.

Despite the representation, the reader must keep in mind that a substitution acts as a mapping and it performs simultaneous substitutions, as in the example above.

In what follows, we say that a name a is *fresh* in a simultaneous substitution σ if a does not appear in σ.

Since the correctness of the following encoding is built upon freshness of names of the term the substitution is applied to, the encoding is parametrized by a process P. Moreover, the encoding is parametrized also by an enumeration Φ of names of π-calculus. Note that this is possible because the set of names is countably infinite. The role of Φ will be make clear after the presentation of the encoding.

[12] In order to make the example clear we consider the term $a.b.c$ rather than a π-calculus process.
[13] When we say that σ maps a to b we mean $\sigma(a) = b$, i.e., the name a will be replaced by b.

Definition 5.22.4 (Encoding of simultaneous substitutions)

$$\begin{aligned} [\![\epsilon]\!]_{(P,\Phi)} &= \epsilon \\ [\![\{b/a\}\cdot\sigma]\!]_{(P,\Phi)} &= [a_f/a][\![\sigma]\!]_{P[a_f/a]}[b/a_f] \end{aligned}$$

Where the name a_f is such that $\Phi(n) = a_f$ for some n that is the least natural number m such that $\Phi(m) = a_f$ and a_f is fresh in P and in $\{b/a\}\cdot\sigma$.

In the encoding above $P[a_f/a]$ is performed by the one-step substitution.

Remark on the enumeration. The encoding could simply pick a name a_f that is fresh in P and in $\{b/a\}\cdot\sigma$. However, this choice makes the encoding able to produce different outputs depending on the choice of fresh names made at any step, i.e., the encoding would not be a function. It is however convenient to avoid the technicalities that arise with dealing with multiple encodings. To this aim, we fix an enumeration of names and pick always the least suitable fresh name (with the property stated in Definition 5.22.4). It is not hard to see that such a suitable name always exists. The encoding $[\![\cdot]\!]_{(P,\Phi)}$ is thus a function.

From now onwards, the choice of the enumeration Φ will always be irrelevant, and we will just write $[\![\sigma]\!]_P$ mentioning only the parameter P.

For the sake of example, the encoding for the simultaneous substituton of the previous example is $[\![\{b/a\}\cdot\{c/b\}\cdot\{a/c\}]\!]_{(a.b.c)} = [a_f/a][b_f/b][c_f/c][a/c_f][c/b_f][b/a_f]$. The reader may want to see these six one-step substitutions applied to the term $a.b.c$:

$$\begin{gathered} (1)\ a.b.c[a_f/a] = a_f.b.c \quad (2)\ a_f.b.c[b_f/b] = a_f.b_f.c \quad (3)\ a_f.b_f.c[c_f/c] = a_f.b_f.c_f \\ (4)\ a_f.b_f.c_f[a/c_f] = a_f.b_f.a\ (5)\ a_f.b_f.a[c/b_f] = a_f.c.a\ (6)\ a_f.c.a[b/a_f] = b.c.a. \end{gathered}$$

The series of one-step substitutions ends up in the term $b.c.a$, as we expected. This is not by chance, the following theorem proves that, thanks to the encoding $[\![\cdot]\!]$, one-step substitutions are able to simulate simultaneous substitutions.

Multiple representations for simultaneous substitutions Before stating the theorem, the reader must know that the mappings of Definition 5.6.4 allow for multiple representations of simultaneous substitutions. For instance, the two substitutions $\{a/b\}\cdot\{c/d\}\cdot\{e/f\}$ and $\{a/b\}\cdot\{e/f\}\cdot\{c/d\}$ represent the same mapping. We equate all of these representations, and when refering to a simultaneous substitution σ, we actually refer to a representative representation of the class of all

the simultaneous substitutions which differs only by permutation of their single substitutions. It is not hard to see that all of these equated representations lead to the same term when applied to a process P.

Theorem 5.22.5 (Correctness of the encoding of simultaneous substitutions) *For all processes P in Π, for all simultaneous substitutions σ, $P\sigma = P[\![\sigma]\!]_P$.*

The proof of Theorem 5.22.5 can be found in Section 5.23. Relying on Theorem 5.22.5, we proceed to prove Theorem 5.22.3. Before embarking on the proof, we state a useful lemma.

The following lemma, whose proof is straightforward, ensures that simultaneous substitutions and one-step substitutions coincide when substituting only one name.

Lemma 5.22.6 *For all $P \in \Pi$, for all names a and b it holds that $P\{a/b\} = P[a/b]$.*

The proof is divided into two cases, given P and $Q \in \Pi$:

1. Soundness: if $P \mathrel{\hat{\leftrightarrow}} Q$ then $P \underline{\leftrightarrow}^{1sO} Q$.
2. Completeness: if $P \underline{\leftrightarrow}^{1sO} Q$ then $P \mathrel{\hat{\leftrightarrow}} Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid P \mathrel{\hat{\leftrightarrow}} Q\}$$

is a one-step open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \mathrel{\hat{\leftrightarrow}} Q$. Consider first Clause 1 of Definition 5.22.1 and assume $P \xrightarrow{\alpha} P'$. Since $\hat{\leftrightarrow}$ ranges over all the simulation substitutions, it ranges also over the simultaneous substitution ι which is the identity over all the names, i.e., such that $P\iota = P$ for all $P \in \Pi$. We have thus that $P\iota \xrightarrow{\alpha} P'$. Since $P \mathrel{\hat{\leftrightarrow}} Q$, we have that there exists a process Q' such that $Q\iota \xrightarrow{\alpha} Q'$, which simply means $Q \xrightarrow{\alpha} Q'$, and $P' \mathrel{\hat{\leftrightarrow}} Q'$. We now have to prove that $P' \mathcal{R} Q'$. This follows from the fact that $P' \mathrel{\hat{\leftrightarrow}} Q'$.

Consider now Clause 2 of Definition 5.22.1, which is about one-step substitutions. It is easy to see that since $\hat{\leftrightarrow}$ ranges over all the simultaneous substitutions, it also ranges over all the simultaneous substitutions that replace only one name with another. By Lemma 5.22.6, these substitutions coincide to their corresponding one-step substitutions.

Completeness: It suffices to show that the relation

assume $P\sigma \xrightarrow{\alpha} P'$. L

$$\mathcal{R} = \{(P, Q) \mid P \underline{\leftrightarrow}^{1sO} Q\}$$

is an open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \underline{\leftrightarrow}^{1sO} Q$. Let us pick a simultaneous substitutions σ and let us consider its encoding as a series of -one-step substitutions $\sigma^* = [\![\sigma]\!]_{(P\|Q)}$ (We use $P \parallel Q$ as parameter in order to make sure that the encoding will pick names that are fresh in both P and Q). Note that if a name is fresh in both P and Q, it is fresh in P and Theorem 5.22.5 applies. The same holds for Q. Let σ^* be $[a_1/a_2][a_3/a_4] \cdot [a_{n-1}, /a_n] \cdot \epsilon$. The first one-step substitution $[a_1/a_2]$ is considered by $\underline{\leftrightarrow}^{1sO}$, and we have that $P[a_1/a_2] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2]$. Now, since $P[a_1/a_2] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2]$, we have that, again by Clause 2 of Definition 5.22.1, the substitution $[a_3, a_4]$ is considered by $\underline{\leftrightarrow}^{1sO}$ from the terms $P[a_1/a_2]$ and $Q[a_1/a_2]$, and $P[a_1/a_2][a_3/a_4] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2][a_3, a_4]$. By iterating the application of Clause 2 of Definition 5.22.1, we can apply to P and Q the complete sequence of one-step substitutions σ^*, ending up with $P\sigma^* \underline{\leftrightarrow}^{1sO} Q\sigma^*$. Now, thanks to the Clause 1 of Definition 5.22.1, since $P\sigma^* \underline{\leftrightarrow}^{1sO} Q\sigma^*$, if $P\sigma^* \xrightarrow{\alpha} P'$ then $Q\sigma^* \xrightarrow{\alpha} Q'$ and $P' \underline{\leftrightarrow}^{1sO} Q'$. By Theorem 5.22.5, $P\sigma^* = P\sigma$, so also $P\sigma$ is such that $P\sigma \xrightarrow{\alpha} P'$. For the same reasons, we have that also $Q\sigma$ is such that $Q\sigma \xrightarrow{\alpha} Q'$. Now, since $P' \underline{\leftrightarrow}^{1sO} Q'$ we have that $P'\mathcal{R}Q'$.

This concludes the proof of Theorem 5.22.3. The statement of Theorem 5.6.6 easily follows from Theorem 5.22.2 and Theorem 5.22.3.

5.23 Simulation of Substitutions by One-Step Substitutions: Proof of Theorem 5.22.5

The proof of Theorem 5.22.5 relies on two lemmas, whose proofs are omitted.

Lemma 5.23.1 *For all $P \in \Pi$, for each simultaneous substitution σ and names a and b, if σ does not map a, b is fresh in P and b does not appear in σ, then it holds that $(P\{b/a\})\sigma = P\{b/a\} \cdot \sigma$.*

The reader must notice that differently from what happens in $(P\{b/a\})\sigma$, in the term $P\{b/a\} \cdot \sigma$ the substitution $\{b/a\}$ is performed simultaneously with the others of σ. The proof of Lemma 5.23.1 is straightforward and omitted.

The following lemma says, instead, that in order to change an atom a into b in a term while performing other substitutions, we can first change the atom a into a fresh new atom a_f and apply simultaneously the further substitutions, and after that, replace the atom a_f with b. We can do this as long as the further substitutions do not change a_f or introduce other occurrences of a_f.

Lemma 5.23.2 *For all $P \in \Pi$, for each simultaneous substitution σ and names a, b and a_f, if σ does not map a and a_f is fresh in P and a_f does not appear in σ, then it holds that $P\{b/a\} \cdot \sigma = (P(\{a_f/a\} \cdot \sigma))\{b/a_f\}$.*

Let P be a process of the π-calculus and σ be a simultaneous substitution. The proof proceeds by induction on the "length" of σ, i.e., the cardinality of the set of atoms over which σ does not act as the identity. Note that by Definition 5.6.4 this set is finite. For what follows, we invite the reader to pay attention to the fact that the two expressions $(P\{a_f/a\})\sigma$ and $P(\{a_f/a\} \cdot \sigma)$ mean different things. The former denotes the substitution σ applied to the term $P\{a_f/a\}$, while the latter denotes the substitution $\{a_f/a\} \cdot \sigma$ applied to the term P.

- Length 0, i.e., $\sigma = \epsilon$: It is easy to see that $P\epsilon = P[\![\epsilon]\!]_P = \epsilon$.
- Length $n > 0$, i.e., $\sigma = \{b/a\} \cdot \sigma'$: Then, $[\![\sigma]\!]_P = [a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f]$, with a_f fresh in P. Now, let us consider the term $P\{a_f/a\}$. By inductive hypothesis, which apply to σ', $(P\{a_f/a\})[\![\sigma']\!]_{P\{a_f/a\}} = (P\{a_f/a\})\sigma'$. By Lemma 5.22.6, we know that $P\{a_f/a\} = P[a_f/a]$, so we have $(P[a_f/a])[\![\sigma']\!]_{P[a_f/a]} = (P\{a_f/a\})\sigma'$. Again, by Lemma 5.22.6, we can add one single additional substitution without affecting the equation, so we have $((P[a_f/a])[\![\sigma']\!]_{P[a_f/a]})[b/a_f] = ((P\{a_f/a\})\sigma')\{b/a_f\}$. Since σ is a mapping, clearly σ' does not map a. We know also that a_f does not appear in σ' and that a_f is fresh in P. We can thus apply Lemma 5.23.1 and have that $(P\{a_f/a\})\sigma' = P\{a_f/a\} \cdot \sigma'$. The equation thus rewrites as $P[a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f] = (P\{a_f/a\} \cdot \sigma')\{b/a_f\}$. Again, since σ' does not map a, a_f does not appear in σ' and a_f is fresh in P, we can apply Lemma 5.23.2 and have that $(P\{a_f/a\} \cdot \sigma')\{b/a_f\} = P\{b/a\} \cdot \sigma'$. We can thus conclude that $P[a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f] = P\{b/a\} \cdot \sigma'$.

5.24 Nominal bisimilarity equates too much in λ-calculus: Proof of Theorem 5.7.4

Before embarking on the proof of Theorem 5.7.4, let us first state some lemmas.

Lemma 5.24.1 (Free atoms of a term after a substitution) *Let T be an NTSS and let M be a term over the signature of T. For all atoms b, it holds that:*

- *If $fa(M) = \Phi$ and $a \notin \Phi$, then $fa(M[b/a]) = \Phi$ (if $M[b/a]$ is defined).*
- *If $fa(M) = \Phi \cup \{a\}$ and $a \notin \Phi$, then $fa(M[b/a]) = \Phi \cup \{b\}$ (if $M[b/a]$ is defined).*

Lemma 5.24.2 (Substituting a free atom with a binding-closed term) *Consider the NTSS of our lazy λ-calculus defined in Section 5.5.1. Let $M \in \mathbb{C}(\Sigma^\lambda)$ and let $fa(M) = \{a\}$, for some atom a. For all terms $M' \in \mathbb{C}(\Sigma^\lambda)$ and $N \in \mathbb{C}(\Sigma^\lambda)^0$, it holds that if $M \xrightarrow{a \overset{T}{\mapsto} N} M'$ then $M' \in \mathbb{C}(\Sigma^\lambda)^0$.*

Lemmas 5.24.1 and 5.24.2 can both be proved by an induction on the structure of M. The proofs are simple and omitted. Note, however, that these proofs rely on the straightforward fact that $a \notin fa([a]M)$ for every atom a and term M.

Lemma 5.24.3 (α-conversions preserve the set of free atoms) *Let T be an NTSS whose set of rules contains the rules for α-conversion transitions as defined in Section 5.4.2. Let M be a term over the signature of T and let $fa(M) = \Phi$. For all terms N, it holds that if $M \to_\alpha N$ then $fa(N) = \Phi$.*

Intuitively, it is easy to see from the rules of Section 5.4.2 that an α-conversion transition does not introduce or delete free atoms in a term. Lemma 5.24.3 can be proved by an induction on the length of the proofs of provable α-conversion transitions. However, such a proof makes use of a technical detail for the case of terms of form $[a].M$. We shall discuss only this case in detail. There are two possible α-conversion transitions from the term $[a].M$. Namely (1) $[a]M \to_\alpha [b]M'$ by rule $(abs1_\alpha)$, with $b\#M$ and $M \xrightarrow{a \overset{A}{\mapsto} b} M'$, and (2) $[a]M \to_\alpha [a]M'$ by rule $(abs2_\alpha)$, with $M \to_\alpha M'$.

Let us consider (1). Let $fa([a].M) = \Phi$. By definition of the set fa, $a \notin \Phi$. We also have that $fa(M) = \Phi$ or $fa(M) = \Phi \cup \{a\}$. Now, By Lemma 5.4.2, which states the correctness of substitution transitions, we know that $M' = M[b/a]$. By Lemma 5.24.1 we can conclude that $fa(M[b/a]) = \Phi$ and $b \notin \Phi$, or $fa(M[b/a]) = \Phi \cup \{b\}$.

Either way, when we abstract on the atom b from $M[b/a]$, we have $fa([b].M[b/a]) = \Phi$.

Let us consider (2). Let $fa([a].M) = \Phi$, it holds again that $a \notin \Phi$ and that $fa(M) = \Phi$, or $fa(M) = \Phi \cup \{a\}$. Now, since the proof length for proving the transition $M \rightarrow_\alpha M'$ is strictly less than the proof length for proving $[a]M \rightarrow_\alpha [a]M'$, by the inductive hypothesis we can conclude that $fa(M') = \Phi$ or $fa(M') = \Phi \cup \{a\}$. Either way, when we abstract on the atom a from M', we have $fa([a].M') = \Phi$.

As a straightforward consequence of Lemma 5.24.3, α-conversion transitions preserve binding-closedness of terms, as stated below.

Lemma 5.24.4 (α-conversions preserve binding-closedness of terms) *Let T be an NTSS with signature Σ and let the set of rules of Σ contain the rules for α-conversion transitions as defined in Section 5.4.2. Let $M \in \mathbb{C}(\Sigma)^0$. For all terms N, it holds that if $M \rightarrow_\alpha N$ then $N \in \mathbb{C}(\Sigma)^0$.*

For any two α-equivalent terms of our formulation of the lazy λ-calculus, it holds that either they both have a normal form or they both do not have a normal form, as stated below.

Lemma 5.24.5 (α-conversions preserve the normal form) *Consider the NTSS of our lazy λ-calculus defined in Section 5.5.1. Let M be a term over Σ^λ and let M have a normal form. For all terms N, it holds that if $M \rightarrow_\alpha N$ then N has a normal form.*

Lemma 5.24.5 follows easily from the fact that the reduction $\rightarrow$ is up to α-equivalence and the transition $\rightarrow_\alpha$ is 'symmetric', as shown in the proof of Theorem 5.4.4 in Section 5.14. These two facts imply that if the transition $M \rightarrow M'$ is provable, then also the transition $N \rightarrow M'$ is provable, by α-converting first N back to M.

Lemma 5.24.6 (Substitutions are always possible on λ-terms) *Consider the NTSS of our lazy λ-calculus defined in Section 5.5.1. Let $M \in \mathbb{C}(\Sigma^\lambda)$. For all atoms a and terms N, it holds that $M \xrightarrow{a \overset{T}{\mapsto} N} M'$ for some term M'.*

Lemma 5.24.6 can be proved by an induction on the length of the proofs of provable substitution transitions. The only delicate point is when performing a substitution $\xrightarrow{a \overset{T}{\mapsto} N}$ in a term M and M contains an abstraction $[b].M_1$ with b not fresh in N. However, as argued in Section 5.5.3, since we set the term-for-atom substitutions of our lazy λ-calculus to be up to α-equivalence, a substitution transition is possible also in this case.

By way of example, consider the provable transition

$$(\lambda([a].\lambda([b].(b\ a)))) \xrightarrow{c \overset{T}{\mapsto} (b\ b)} (\lambda([a].\lambda([d].(d\ a)))),$$

where d is fresh in $(b\ a)$. In this case an α-conversion takes place before the application of rule $(abs1_{\mathrm{Ts}})$.

This point is not shown formally here. However, it follows exactly the same reasoning employed in case 3 of the enumeration that contains the proofs of a few cases for Lemma 5.24.7 below.

Lemma 5.24.7 (Substitutions are 'ineffective' on binding-closed terms) *Consider the NTSS of our lazy λ-calculus defined in Section 5.5.1. Let $M \in \mathbb{C}(\Sigma^{\lambda})^0$ and let b be an atom fresh in M. For all terms N and M', it holds that if $M \xrightarrow{b \overset{T}{\mapsto} N} M'$ then $M \rightarrow_{\alpha} M'$.*

Intuitively, it is easy to see that the substitution transitions defined by the rules of Section 5.4.1 can only substitute free atoms in a term. Since a binding-closed term contains no free atom, a substitution ends up in the same term up to α-equivalence.

Lemma 5.24.7 can be proved by an induction on the length of the proofs of provable substitution transitions. However, the cases involving an abstraction $[a].M$ are quite delicate and we wish to discuss them in detail.

1. Let us consider the term $[a]M$ and the substitution transition $\xrightarrow{b \overset{T}{\mapsto} N}$ where $b \neq a$ and a is fresh in N. By the proviso of the lemma, we have also that b is fresh in $[a]M$. In this case we can apply only the rule $(abs1_{\mathrm{Ts}})$, as $(abs2_{\mathrm{Ts}})$ requires that $a = b$. The rule $(abs1_{\mathrm{Ts}})$ is instanciated as follows.

$$\frac{M \xrightarrow{b \overset{T}{\mapsto} N} M' \quad a\#N \quad a \neq b}{[a]M \xrightarrow{b \overset{T}{\mapsto} N} [a]M'}$$

   Notice that the premises $a\#N$ and $a \neq b$ are satisfied. Moreover, since b is fresh in $[a]M$, then $b \notin fa(M)$. Indeed, the term M may have only the atom a as free, if any. Since b is fresh in M, and since the proof length for proving the transition $M \xrightarrow{b \overset{T}{\mapsto} N} M'$ is strictly less than the proof length for proving $[a]M \xrightarrow{b \overset{T}{\mapsto} N} [a]M'$, the inductive hypothesis the inductive hypothesis applies

and $M \rightarrow_\alpha M'$. Therefore, the transition proved above is $[a]M \xrightarrow{b \overset{T}{\mapsto} N} [a]M'$, with $[a]M \rightarrow_\alpha [a]M'$ (by rule $(abs2_\alpha)$).

2. Let us consider the term $[a].M$ and the substitution transition $\xrightarrow{a \overset{T}{\mapsto} N}$ (it is irrelevant whether a is fresh in N or not). In this case we cannot apply the rule $(abs1_{\text{Ts}})$ above, as the premise $a \neq b$ is not satisfied. However, we can apply the rule $(abs2_{\text{Ts}})$ and prove the transition $[a]M \xrightarrow{a \overset{T}{\mapsto} N} [a]M$, for which it clearly holds that $[a]M \rightarrow_\alpha [a]M$.

3. The case discussed in this item is related to the case pointed out for Lemma 5.24.6 above. Let us consider the term $[a].M$ and the substitution transition $\xrightarrow{b \overset{T}{\mapsto} N}$ where $b \neq a$ and a is *not* fresh in N. By the proviso of the lemma, we have also that b is fresh in $[a]M$. In this case we cannot apply the rule $(abs1_{\text{Ts}})$, as it requires that $a\#N$. Also, we cannot apply the rule $(abs2_{\text{Ts}})$, as it requires that $a = b$. However, since the term-for-atom substitution transitions are set to be up to α-equivalence, we can pick an atom c that is fresh in N and instanciate the rule $(b \overset{T}{\mapsto} N \cdot \text{upTo}\alpha)$ as follows.

$$\frac{[a]M \rightarrow_\alpha [c]M' \quad [c]M' \xrightarrow{b \overset{T}{\mapsto} N} [d]M''}{[a]M \xrightarrow{b \overset{T}{\mapsto} N} [d]M''}.$$

Now, by Lemma 5.24.3, since $[a]M \rightarrow_\alpha [c]M'$ we have that $fa([a]M) = fa([c]M')$. This means that since b is fresh in $[a]M$ then b is fresh also in $[c]M'$. Given this fact, and also since the proof length for proving the transition $[c]M' \xrightarrow{b \overset{T}{\mapsto} N} [d]M''$ is strictly less than the proof length for proving $[a]M \xrightarrow{b \overset{T}{\mapsto} N} [d]M''$, the inductive hypothesis applies and we have that $[c]M' \rightarrow_\alpha [d]M''$. Since $[a]M \rightarrow_\alpha [c]M'$ is provable and also $[c]M' \rightarrow_\alpha [d]M''$ is provable, we can use the rule $(\alpha \cdot \text{upTo}\alpha)$, (basically closing by transitivity), and prove the transition $[a]M \rightarrow_\alpha [d]M''$ as required.

The reader may benefit also from an informal intuition on the dynamics at play in the cases above. The enumeration below contains useful examples and will serve this purpose. Note that, each number of the enumeration for the examples below corresponds to the number of the enumeration of the cases proved above. In what follows, atoms with distinct names are considered different atoms.

1. $(\lambda([a].\lambda([b].(b\ a)))\ \overset{c\overset{T}{\mapsto}(d\ d)}{\longrightarrow}\ (\lambda([a].\lambda([b].(b\ a)))$, employing the rule ($\lambda$Ts) with $(abs1_{\mathrm{Ts}})$.

2. $(\lambda([a].\lambda([b].(b\ a)))\ \overset{a\overset{T}{\mapsto}(d\ d)}{\longrightarrow}\ (\lambda([a].\lambda([b].(b\ a)))$, employing the rule ($\lambda$Ts) with $(abs2_{\mathrm{Ts}})$. Employing the same rules we can prove also $(\lambda([a].\lambda([b].(b\,a)))\ \overset{a\overset{T}{\mapsto}(b\ b)}{\longrightarrow}$ $(\lambda([a].\lambda([b].(b\ a)))$.

3. $(\lambda([a].\lambda([b].(b\ a)))\ \overset{c\overset{T}{\mapsto}(b\ b)}{\longrightarrow}\ (\lambda([a].\lambda([d].(d\ a)))$, where d is fresh in $(b\ a)$. This case is related to Lemma 5.24.6.

Lemma 5.24.8 (Reductions preserve binding-closedness and the normal form) *Consider the NTSS of our lazy λ-calculus defined in Section 5.5.1. Let $M \in \mathbb{C}(\Sigma^{\lambda})^0$ have a normal form. For all terms M', it holds that if $M \to M'$ then $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ and M' has a normal form.*

Lemma 5.24.8 can be proved by an induction on the length of the proofs of provable transitions $\to$. We single out only the case when the rule (*app*) is applied. Let us assume that $(M\,N) \to M'''$ is provable by the rule (*app*) instanciated as follows.

$$\frac{M \to \lambda([a]M') \quad M' \overset{a\overset{T}{\mapsto}N}{\longrightarrow} M'' \quad M'' \to M'''}{(M\,N) \to M'''}\ (\mathrm{app})$$

Now, since $(M\ N) \in \mathbb{C}(\Sigma^{\lambda})^0$, we have that $M \in \mathbb{C}(\Sigma^{\lambda})^0$ and $N \in \mathbb{C}(\Sigma^{\lambda})^0$. As the transition $M \to \lambda([a]M')$ is provable, M has a normal form. Since also $M \in \mathbb{C}(\Sigma^{\lambda})^0$ and the fact that the proof length for proving the transition $M \to \lambda([a]M')$ is strictly less than the proof length for proving $(M\ N) \to M'''$, the inductive hypothesis applies and we have that $\lambda([a]M') \in \mathbb{C}(\Sigma^{\lambda})^0$. This means that $fa(M') = \emptyset$ or $fa(M') = \{a\}$. If $fa(M') = \emptyset$, then $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ and by Lemmas 5.24.7 and 5.24.3 we can conclude that after the transition $M' \overset{a\overset{T}{\mapsto}N}{\longrightarrow} M''$ we have that $M'' \in \mathbb{C}(\Sigma^{\lambda})^0$ (by applying also Lemma 5.24.4 to M''). If $fa(M') = \{a\}$, since $N \in \mathbb{C}(\Sigma^{\lambda})^0$, we can apply Lemma 5.24.2 and have that $M'' \in \mathbb{C}(\Sigma^{\lambda})^0$. Either way, we have that $M'' \in \mathbb{C}(\Sigma^{\lambda})^0$. Now, since the transition $M'' \to M'''$ is provable, M'' has a normal form. Since also $M'' \in \mathbb{C}(\Sigma^{\lambda})^0$ and the fact that the proof length for proving the transition $M'' \to M'''$ is strictly less than the proof length for proving $(M\,N) \to M'''$, the inductive hypothesis applies and we have that $M''' \in \mathbb{C}(\Sigma^{\lambda})^0$ and has a normal form.

The rest of the proof proceeds following standard lines. Note, however, that the proof for transitions $M \to M'$ up to α-conversion, i.e. an α-conversion takes

place first, makes use of Lemmas 5.24.4 and 5.24.5, which state that α-conversions preserve binding-closedness and 'having a normal form', respectively.

We are now ready to prove Theorem 5.7.4. To this end, it is sufficient to show that the relation

$$\mathcal{R} = \{(M, N) \mid M, N \in \mathbb{C}(\Sigma^{\lambda})^0 \textit{ and } M, N \textit{ have normal form}\}$$

is a nominal bisimulation.

Notice first of all that $\mathcal{R}$ is symmetric. Assume now that $M \,\mathcal{R}\, N$. There are three types of transitions that we need to consider within the bisimulation game of Definition 5.6.2. Namely, (1) substitutions transitions, (2) α-conversion transitions and (3) transitions performed by the reduction step $\rightarrow$.

Let us consider first the case (1). Let us pick an atom a and a term P and assume that $M \xrightarrow{a \overset{T}{\mapsto} P} M'$, for some term M'. Since $M \in \mathbb{C}(\Sigma^{\lambda})^0$, by Lemma 5.24.7 we know that $M \rightarrow_{\alpha} M'$. Since $M \in \mathbb{C}(\Sigma^{\lambda})^0$ and M has a normal form, by Lemmas 5.24.4 and 5.24.5 we can conclude that $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ and also that M' has a normal form. Now, since $N \in \mathbb{C}(\Sigma^{\lambda})^0$, by Lemma 5.24.6 substitutions are always possible and we have that $N \xrightarrow{a \overset{T}{\mapsto} P} N'$ for some term N'. By Lemma 5.24.7, we also know that $N \rightarrow_{\alpha} N'$. Again, since $N \in \mathbb{C}(\Sigma^{\lambda})^0$ and N has a normal form, by Lemmas 5.24.4 and 5.24.5 we can conclude that $N' \in \mathbb{C}(\Sigma^{\lambda})^0$ and also that N' has a normal form. We therefore have that $M' \,\mathcal{R}\, N'$.

Let us consider now the case (2) and assume that $M \rightarrow_{\alpha} M'$, for some term M'. Using Lemmas 5.24.4 and 5.24.5, we know that $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ and that M' has a normal form. Now, we can use the rule (id_{α}) in order to prove the transition $N \rightarrow_{\alpha} N$. Since $N \in \mathbb{C}(\Sigma^{\lambda})^0$ and N has a normal form, we have therefore that $M' \,\mathcal{R}\, N$.

Let us consider now the case (3) and assume that $M \rightarrow M'$, for some term M'. By Lemma 5.24.8 we know that $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ and that M' has a normal form. Now, since N has a normal form, we have that we can prove a transition $N \rightarrow N'$, for some term N'. Moreover, by Lemma 5.24.8 we have that $N' \in \mathbb{C}(\Sigma^{\lambda})^0$ and also that N' has a normal form. We have therefore that $M' \,\mathcal{R}\, N'$. This concludes the proof.

Chapter 6

Conclusions

> *Don't let it end like this. Tell them I said something.*
> Pancho Villa.

SOS is one of the most natural and successful ways for providing programming and specification languages with a formal semantics. The meta-theory of SOS has been investigated for over 20 years and has led to many foundational results from which the community can benefit.

This thesis has presented some contributions to the meta-theory of SOS. The contributions are not specific to a single subject. They rather attack different research problems.

In Chapter 2 we proposed rule-matching bisimilarity, a method for establishing the soundness of equations between open terms constructed using operations whose semantics is specified by rules in the GSOS format of Bloom, Istrail and Meyer [44]. We proved that rule-matching bisimilarity is a sound proof method for showing the validity of equations with respect to bisimilarity. The method is not complete. However, we show by means of examples that rule-matching bisimilarity is expressive enough to establish the validity of relevant equations from the literature.

An extension of the proof method to the setting of GSOS languages with predicates is also offered.

Proofs of validity of equations (modulo bisimilarity) are often lengthy, work-intensive and need to be carried out for many equations and languages. Automated methods devoted to this task are thus desirable, especially if they are

applicable in a general fashion, as in the context of rule formats, rather than confined to specific calculi.

We believe that rule-matching bisimilarity represents a good tool for the mechanization of the task. The reader should also bear in mind that the GSOS rule format and its extension with predicates, although constrained on the permissible shapes of their rules, are still expressive enough to formulate many languages. For instance, most process algebras can be specified within them.

We point out some future research direction concerning the theory of rule-matching bisimilarity.

- **Rensink's HP-bisimilarity**: Rule-matching bisimilarity is inspired by de Simone's FH-bisimilarity [52]. In [125], Rensink presented a natural sharpening of de Simone's FH-bisimilarity, the hypothesis preserving bisimilarity. His extension is orthogonal to ours, and it adds to FH-bisimilarity the capability to store some information about the variable transitions during the computation. Adding this feature to our method would lead to a more powerful rule-matching equivalence, in particular the validity of the equation in Example 2.5.4 of Section 2.5, which witnesses the incompleteness of the rule-matching bisimilarity, would be provable. We leave this sharpening as future work.

- **Extension to more expressive rule formats**: It would be worthwhile to extend our method to rule formats that are more expressive than the GSOS format, for example NTyft/NTyxt [66] and Panth [150], just to name a few. The main challenge in this context amounts in providing a corresponding Ruloid Theorem for the rule format considered and also a logic that is capable of reasoning about the satisfiability of its premises. Moreover, for mechanization purposes, the entailment of this logic, as defined in Section 2.4, must be decidable.

- **Completeness for restricted classes of languages**: Although we show that the rule-matching bisimilarity is incomplete within the full GSOS format, in Section 2.7 we provide some restricted classes within GSOS for which our method is a complete proof technique for establishing equality between open terms. Admittedly, those classes are not quite expressive and as future work, we plan to investigate complete classes that are more interesting than the ones proposed.

- **Robustness of equations**: Sound equations may become unsound after an operationally conservative extension; for examples see [107] , [103] and Example 2.5.6 on page 37. It is thus desirable to provide criteria under which the validity of equations is instead preserved. The main benefit is that it prevents the user from repeating proofs in the context of the new extended semantics.

  We provide some criteria along this line in the context of rule-matching bisimilarity. Theorems 2.5.8 and 2.8.11 offer some conditions guaranteeing that the equations proved by our method are preserved in disjoint extensions of the original language. It would be worthwhile to provide more results of this kind. To this end, we may benefit largely from the work in [103], where the authors address the topic of the robustness of equations in a reasoned account. Especially, we may benefit from [102], a longer version of the mentioned paper, which contains also some considerations about the rule-matching bisimilarity in this context.

- **Implementation**: We plan on working on an implementation of a prototype checker for rule-matching bisimilarity.

In Chapters 3 and 4, we provide rule formats that tackle two basic algebraic laws that have not been addressed before in the context of rule formats, i.e., the existence of zero elements and the distributivity law. We showed that several classical examples from the literature indeed fit the formats, witnessing their applicability and expressiveness. In Chapter 3, we also reformulate one of the rule formats for zero elements in order to address unit elements. A rule format for unit elements has been already addressed in an earlier contribution, [22]. It turns out, however, that the two formats are incomparable.

Mechanizing the rule formats in a tool-set is a long-term goal of research on SOS rule formats. We believe that the formats presented in Chapters 3 and 4 are strong candidates for mechanization for checking the addressed algebraic laws.

With the contributions contained in Chapters 3 and 4 of this thesis, the meta-theory of SOS tackles all the basic algebraic laws in the context of rule formats. Namely, we currently have rule formats for

- commutativity in [110],
- associativity in [49],
- idempotence in [4],

- unit elements in [22] and in Chapter 3,
- zero elements in Chapter 3, and
- distributivity in Chapter 4.

In a sense, the contributions presented in Chapters 3 and 4 complete the picture insofar the basic algebraic laws is concerned.

However, a substantial amount of work on rule formats still has to be done. We point out some future research.

- **Improvement of existing formats**: To begin with, there is room for improvements of the mentioned rule formats. We just name two examples. The associativity rule format in [49] can be extended in order to be applied to SOS semantics that use negative premises. This would allow us to recognize associativity laws that are not captured so far by rule formats. An example is the associativity of the sequencing operator ; that is discussed in Section 2.6[1].

  Also, the rule formats for distributivity presented in Chapter 4 deal with operators $\oplus$ that must be choice-like. It is worthwhile to extend the rule formats in order to address a larger class of operators.

- **Rule formats for other properties**: Some basic properties of transition systems have not been addressed yet by rule formats. We limit ourselves to mention the most prominent example: the *confluence* property. The notion of confluence arises in various forms and with different motivations in several areas within the theory of computation, see, for instance, [33, 96]. Intuitively, confluence guarantees determinacy in computation by requiring that, of any two possible computation steps, the occurrence of one will never preclude the other. Despite the fact that this property occurs in many contexts and has been subject of extensive investigation, a rule format guaranteeing confluence has not been offered yet and it is part of our future work.

- **Impossibility results**: In Section 4.7 of Chapter 4 we offer some impossibility results concerning the validity of the distributivity law. Unlike previous results about rule formats for algebraic properties, these results allow one to recognize when a certain algebraic law is guaranteed *not* to hold. When designing operational specifications for operators that are intended to satisfy

[1] In Section 2.6 we establish the validity of this associativity law using the rule-matching bisimilarity.

a certain algebraic law, a language designer might benefit from considering also these kinds of negative results.

To our knowledge this type of result does not have any precursor in the field of rule formats. Hitherto, all rule formats aimed at providing sufficient conditions for establishing semantic properties, whereas the above-mentioned results are the first ones that offer *necessary syntactic conditions* for some semantic property to hold.

It would be worthwhile to investigate analogous impossibility results also in the context of other properties. In particular in the context of congruence formats and also concerning the other algebraic laws.

In Chapter 5 we propose Nominal SOS, a formal framework for the handling of names and binders in SOS. We treat some basic notions that are common in nominal calculi, such as α-conversion and substitution. These notions are usually left out of the level of the semantics and they are treated as operations at the meta-level. As our design choice, Nominal SOS does not come equipped with such notions. We show, however, that they can be naturally captured within the Nominal SOS framework as ordinary transitions. The language specifier has the possibility to define and explore, for instance, other types of substitution or equivalence of terms. Moreover, since α-conversion and substitutions are transitions just like any other, they can be the subject of meta-theorems based on the shape of rules that may be developed in the future.

We specify two of the most prominent examples of nominal calculi, namely the lazy λ-calculus and the early π-calculus, in Nominal SOS and show that our specifications coincide operationally with the original definitions of [2] and [132], respectively. These examples show the expressiveness and perhaps the naturalness of the framework. From the rules that formulate the early π-calculus we can see that freshness premises play a role. Indeed, the novelty of having freshness tests in rules is not only useful in modelling in a direct way meta-level operations such as substitutions and α-conversion, but it is also useful in the modelling of specific features of languages.

Finally, we define a notion of nominal bisimilarity naturally arising from our framework. We show that in the case of the π-calculus our notion coincides with the well-known open bisimilarity, [132, 131].

The meta-theory of SOS is by now a mature field. However, relatively little meta-theory has been done in the context of calculi with binders. We believe that

Nominal SOS represents a good candidate in order to carry out a systematic study of the meta-theory for those phenomena that are specifically related to calculi with binders.

Nominal SOS is close enough to the framework of the ordinary SOS and we believe that this similarity with ordinary SOS will have the following main benefits.

- We will be able to lift/adapt already existing meta-theory to the context of binders with relative little effort.
- The content of the meta-theorems and the line of investigation will resemble closely those achieved so far for ordinary SOS, and with which SOS users are familiar.
- Moreover, thanks to the nominal approach, we hopefully have an intuitive and familiar language with which we could explain why certain calculi afford a property while others do not, for instance by means of presence/absence of freshness premises, or of a syntactic discipline in the use of them or of other related nominal concepts.

The overarching aim is to develop the framework of Nominal SOS in a way that is comparable to that of the standard theory of SOS, as surveyed in, e.g., [18, 109], and to hopefully establish Nominal SOS as a framework of reference for the study and the development of the theory of languages with first-class notions of names and binders. More specifically, the main goals of our future work are as follows.

- **Application of Nominal SOS**: We intend to provide further evidence that Nominal SOS is expressive enough to capture the original semantics of nominal calculi, such as variants of the π-calculus and its higher-order version [130], the psi-calculi [36] and the object calculi [1], and to prove formally the correspondence between the presentation in terms of Nominal SOS and the original ones. Also, we plan to address different notion of equivalence betweens terms. Just to name a few examples, it would be worth defining within Nominal SOS a notion that is the analogous of the applicative bisimilarity in the context of the λ-calculus, [2], and investigating the relation between the notion of nominal bisimilarity and the applicative bisimilarity. Also, it is part of our future work to adapt the definition of nominal bisimilarity in order to coincide with the open bisimilarity with distinctions, [132, 131], in the context of π-calculus terms.

- **Extending the framework of Nominal SOS**: Chapter 5 should be considered as containing the basic developments of the framework of Nominal SOS. We are aware of a few possible extensions of the framework that would be worth adding. For instance, it seems natural to consider also the possibility to state negative freshness premises in rules. It would be also worth adding the possibility to write nominal terms of the form $([x_{\mathbb{A}}]t_{\sigma})_{[\mathbb{A}]\sigma}$. Namely, terms can abstract also on a variable that ranges over atoms rather than on a concrete atom. This feature should be accompanied by adding premises of the form $x\#t$, with x a variable of atom sort. As argued in Section 5.9.1, this extension seems useful in some contexts in order to avoid the replication of certain rules for all atoms. For instance, the λ-calculus would be finitely formalized in this extension of Nominal SOS. Another research line consists in providing to the user a suitable language capable of defining the meaning of the binders involved in the language at hand.

- **Developing the meta-theory of Nominal SOS**: We plan to develop the meta-theory of Nominal SOS, for example by providing congruence formats for behavioural semantics in the context of calculi with binders, possibly generalizing those proposed in [153] and [56], for instance. Also, we mentioned above that *confluence* is an important property that has not been tackled yet by the theory of SOS in the context of rule formats. Many important confluence results stem from the realm of calculi equipped with binders; the reader may indeed think of the λ-calculus and its variants. It would be thus desirable to provide rule formats guaranteeing the confluence property within the framework of Nominal SOS. Meta-theory over Nominal SOS can be carried out also for all those phenomena that are specifically related to binders. For instance, it would be worth providing rule formats guaranteeing that the late and early bisimilarity coincide, see [132].

- **Lifting/adapting results from the ordinary SOS**: We expect to extend a wealth of classic SOS meta-results and techniques to the framework of Nominal SOS.

- **Implementation**: We plan on providing tool support for Nominal SOS.

Bibliography

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, New York, 1996. [cited at p. 7, 194, 242]

[2] Samson Abramsky. The lazy lambda calculus. In D.A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, Reading, Mass., 1990. [cited at p. 16, 156, 160, 171, 172, 182, 194, 204, 241, 242]

[3] Samson Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, 1991. [cited at p. 11]

[4] Luca Aceto, Arnar Birgisson, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. Rule formats for determinism and idempotency. In *Proceedings of the 3rd International Conference on Fundamentals of Software Engineering (FSEN'09)*, Lecture Notes in Computer Science, Kish Island, Iran, 2009. Springer-Verlag, Berlin, Germany. [cited at p. 11, 14, 57, 68, 110, 125, 132, 135, 239]

[5] Luca Aceto, Bard Bloom, and Frits Vaandrager. Turning sos rules into equations. *Information and Computation*, 111:1–52, May 1994. [cited at p. 13, 24, 25, 44, 111, 122, 123]

[6] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. Axiomatizing gsos with predicates. In *Proceedings of the 8th Workshop on Structural Operational Semantics 2011 (SOS 2011)*, volume 62 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–15, 2011. [cited at p. 13]

[7] Luca Aceto, Matteo Cimini, and Anna Ingolfsdottir. A bisimulation-based method for proving the validity of equations in GSOS languages. In *Proceedings of the 6th Workshop on Structural Operational Semantics 2009 (SOS 2009), August 31, 2009, Bologna (Italy)*, volume 18 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16, 2010. [cited at p. 16, 68, 87, 88, 89, 100, 142]

[8] Luca Aceto, Matteo Cimini, and Anna Ingolfsdottir. Proving the validity of equations in gsos languages using rule-matching bisimilarity, 2011. to appear in *Mathematical Structures in Computer Science*. [cited at p. 16]

[9] Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Mohammad Reza Mousavi, and Michel A. Reniers. Sos rule formats for zero and unit elements. *Theoretical Computer Science*, 412(28):3045–3071, 2011. [cited at p. 16]

[10] Luca Aceto, Matteo Cimini, Anna Ingolfsdottir, Mohammadreza Mousavi, and Michel A. Reniers. Rule formats for distributivity. Technical Report CSR-10-16, TU/Eindhoven (2010). [cited at p. 16]

[11] Luca Aceto, Matteo Cimini, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. On rule formats for zero and unit elements. In *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVI), Ottawa, Canada*, volume 265 of *Electronic Notes in Theoretical Computer Science*, pages 145–160. Elsevier B.V., The Netherlands, 2010. [cited at p. 16, 110]

[12] Luca Aceto, Matteo Cimini, Anna Ingolfsdottir, Mohammadreza Mousavi, and Michel A. Reniers. Rule formats for distributivity. In *Proceedings of the 5th International Conference on Language and Automata Theory and Applications (LATA 2011)*, volume 6638 of *Lecture Notes in Computer Science*, pages 79–90. Springer-Verlag, 2011. [cited at p. 16]

[13] Luca Aceto, Wan Fokkink, Anna Ingolfsdottir, and Bas Luttik. CCS with Hennessy's merge has no finite equational axiomatization. *Theoretical Computer Science*, 330(3):377–405, 2005. [cited at p. 111]

[14] Luca Aceto, Wan Fokkink, Anna Ingolfsdottir, and Bas Luttik. Finite equational bases in process algebra: Results and open questions. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *Lecture Notes in Computer Science*, pages 338–367. Springer, 2005. [cited at p. 19, 111]

[15] Luca Aceto, Wan Fokkink, Anna Ingolfsdottir, and Bas Luttik. A finite equational base for ccs with left merge and communication merge. *ACM Transactions on Computational Logic*, 10:6:1–6:26, January 2009. [cited at p. 20, 111]

[16] Luca Aceto, Wan Fokkink, Anna Ingolfsdottir, and Sumit Nain. Bisimilarity is not finitely based over BPA with interrupt. *Theoretical Computer Science*, 366(1–2):60–81, 2006. [cited at p. 111]

[17] Luca Aceto, Wan Fokkink, and Chris Verhoef. Conservative extension in structural operational semantics. *Bulletin of the European Association for Theoretical Computer Science*, 69:504–524, 1999. [cited at p. 11]

[18] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 1999. [cited at p. 10, 13, 21, 26, 67, 69, 109, 112, 124, 156, 157, 158, 193, 194, 242]

[19] Luca Aceto and Anna Ingolfsdottir. Cpo models for gsos languages - part i: Compact gsos languages. *Information and Computation*, 129, 1994. [cited at p. 11]

[20] Luca Aceto, Anna Ingólfsdóttir, Bas Luttik, and Paul van Tilburg. Finite equational bases for fragments of ccs with restriction and relabelling. In *Proceedings of the 5th IFIP International Conference On Theoretical Computer Science (IFIP TCS 2008)*, volume 273 of *IFIP*, pages 317–332, 2008. [cited at p. 20]

[21] Luca Aceto, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. Algebraic properties for free! *Bulletin of the EATCS*, 99:81–104, October 2009. [cited at p. 67, 110, 126, 139]

[22] Luca Aceto, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. Rule formats for unit elements. In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, Lecture Notes in Computer Science, Spindleruv Mlyn, Czech Republic, 2010. Springer-Verlag, Berlin, Germany. [cited at p. 11, 57, 68, 73, 76, 77, 85, 86, 94, 98, 100, 104, 110, 239, 240]

[23] The Coq Proof Assistant. http://coq.inria.fr/. [cited at p. 192]

[24] Jos C. M. Baeten, Twan Basten, and Michel A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. [cited at p. 21, 80, 109, 111, 121, 136, 137]

[25] Jos C. M. Baeten and Jan A. Bergstra. Process algebra with a zero object. In *Proceedings of the Theories of Concurrency: Unification and Extension (CONCUR '90)*, volume 458 of *Lecture Notes in Computer Science*, pages 83–98, London, UK, 1990. Springer-Verlag. [cited at p. 53, 82]

[26] Jos C. M. Baeten and Jan A. Bergstra. Mode transfer in process algebra. Technical Report Report CSR 00–01, Eindhoven University of Technology, 2000. [cited at p. 122]

[27] Jos C. M. Baeten, Jan A. Bergstra, and Jan W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986. [cited at p. 123, 144]

[28] Jos C. M. Baeten and Sjouke Mauw. Delayed choice: An operator for joining Message Sequence Charts. In *Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques (FORTE 1994)*, volume 6 of *IFIP Conference Proceedings*, pages 340–354. Chapman & Hall, 1995. [cited at p. 122]

[29] Jos C. M. Baeten and Cornelis A. Middelburg. *Process Algebra with Timing*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin, 2002. [cited at p. 80, 156]

[30] Jos C. M. Baeten and Frits W. Vaandrager. An algebra for process creation. *Acta Informatica*, 29(4):303–334, 1992. [cited at p. 41]

[31] Jos C. M. Baeten and Chris Verhoef. A congruence theorem for structured operational semantics with predicates. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 477–492. Springer-Verlag, Berlin, Germany, 1993. [cited at p. 10]

[32] Jos C. M. Baeten and Erik P. de Vink. Axiomatizing gsos with termination. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS '02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 583–595, London, UK, 2002. Springer-Verlag. [cited at p. 13, 47, 93]

[33] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics, Revised edition*. North Holland, 1984. [cited at p. 159, 160, 172, 185, 240]

[34] Henk. P. Barendregt. *Lambda calculi with types*, pages 117–309. Oxford University Press, Inc., New York, NY, USA, 1992. [cited at p. 190]

[35] Falk Bartels. Gsos for probabilistic transition systems. *Electronic Notes in Theoretical Computer Science*, 65(1):29–53, 2002. [cited at p. 13]

[36] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48, Washington, DC, USA, 2009. IEEE Computer Society. [cited at p. 194, 242]

[37] Jan A. Bergstra and Jan W. Klop. Fixedpoint semantics in process algebra. Technical Report IW 206/82, Center for Mathematics, Amsterdam, The Netherlands, 1982. [cited at p. 68, 81, 121, 137]

[38] Jan A. Bergstra and Jan W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984. [cited at p. 109, 111, 121]

[39] Jan A. Bergstra and Cornelis A. Middelburg. Preferential choice and coordination conditions. *Journal of Logic and Algebraic Programming*, 70(2):172–200, 2007. [cited at p. 84]

[40] Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *LICS*, pages 153–164, 1998. [cited at p. 10, 166]

[41] George David Birkhoff. A set of postulates for plane geometry, 1932. Annals of Mathematics 33. [cited at p. 9]

[42] Bard Bloom. Many meanings of monosimulation: Denotational. operational and logical characterizations of a notion of simulation of concurrent processes, 1991. Unpublished manuscript. [cited at p. 11]

[43] Bard Bloom, Wan Fokkink, and Rob J. van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Transactions on Computational Logic*, 5(1):26–78, 2004. [cited at p. 29]

[44] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '88, pages 229–239, New York, NY, USA, 1988. ACM. [cited at p. 10, 11, 13, 16, 20, 21, 24, 25, 28, 29, 48, 68, 72, 237]

[45] Christiano de O. Braga, Edward H. Haeusler, José Meseguer, and Peter D. Mosses. Mapping modular sos to rewriting logic. In *Proceedings of the 12th international conference on Logic based program synthesis and transformation*, LOPSTR'02, pages 262–277, Berlin, Heidelberg, 2003. Springer-Verlag. [cited at p. 156]

[46] Ed Brinksma. A tutorial on LOTOS. In *Proceedings of the IFIP WG6.1 Fifth International Conference on Protocol Specification, Testing and Verification*, pages 171–194. North-Holland Publishing Co., 1985. [cited at p. 122]

[47] Roberto Bruni, David de Frutos-Escrig, Narciso Marti-Oliet, and Ugo Montanari. Bisimilarity congruences for open terms and term graphs via tile logic. In *Proceedings of CONCUR 2000 - Concurrency Theory, 11th International Conference*, volume 1877 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000. [cited at p. 20, 56]

[48] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS 1998)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, April 1998. [cited at p. 7]

[49] Sjoerd Cranen, MohammadReza Mousavi, and Michel A. Reniers. A rule format for associativity. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 447–461, Toronto,Canada, 2008. Springer-Verlag, Berlin, Germany. [cited at p. 11, 57, 68, 92, 110, 126, 239, 240]

[50] Martin Davis. *Engines of Logic: Mathematicians and the Origin of the Computer*. W. W. Norton & Co., Inc., New York, NY, USA, 2001. [cited at p. 8]

[51] Robert de Simone. *Calculabilité et Expressivité dans l'Algèbre de Processus Parallèles* MEIJE. Thèse de 3^e cycle, Univ. Paris 7, 1984. [cited at p. 34, 36, 43]

[52] Robert de Simone. Higher-level synchronising devices in meije-sccs. *Theoretical Computer Science*, 37:245–267, 1985. [cited at p. 16, 20, 34, 36, 38, 41, 42, 43, 54, 56, 124, 238]

[53] Guillame Doumenc, Eric Madelaine, and Robert De Simone. Proving process calculi translations in ECRINS. Technical Report RR1192, INRIA, 1990. [cited at p. 34]

[54] Maribel Fernandez and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007. [cited at p. 161]

[55] William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core cml. *Journal of Functional Programming*, 8(5):447–491, 1998. [cited at p. 7]

[56] Marcelo Fiore and Sam Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages

49–58, Washington, DC, USA, 2006. IEEE Computer Society. [cited at p. 184, 194, 243]

[57] Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 93–104. IEEE Computer Society, Los Alamiots, CA, USA, 2001, 2001. [cited at p. 184]

[58] Wan Fokkink, Rob J. van Glabbeek, and Paulien de Wind. Compositionality of hennessy-milner logic by structural operational semantics. *Theoretical Computer Science*, 354(3):421–440, 2006. [cited at p. 13, 29]

[59] Wan Fokkink and Chris Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation*, 146:24–54, October 1998. [cited at p. 11, 24, 184]

[60] Wan Fokkink and Thuy Duong Vu. Structural operational semantics and bounded nondeterminism. *Acta Informatica*, 39(6-7):501–516, 2003. [cited at p. 11]

[61] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999. [cited at p. 156, 186, 187]

[62] Andrew Gacek. The abella interactive theorem prover (system description). In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR '08, pages 154–161, Berlin, Heidelberg, 2008. Springer-Verlag. [cited at p. 189]

[63] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 33–44, Washington, DC, USA, 2008. IEEE Computer Society. [cited at p. 189]

[64] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Reasoning in Abella about structural operational semantics specifications. In A. Abel and C. Urban, editors, *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008)*, number 228 in Electronic Notes in Theoretical Computer Science, pages 85–100, 2008. [cited at p. 184, 189, 190]

[65] D. Plotkin Gordon. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004. [cited at p. 21, 67, 109, 155]

[66] Jan F. Groote. Transition system specifications with negative premises (extended abstract). In *CONCUR '90: Proceedings on Theories of concurrency : unification and extension*, pages 332–341, New York, NY, USA, 1990. Springer-Verlag New York, Inc. [cited at p. 10, 238]

[67] Jan F. Groote, Michel A. Reniers, Jos J. Van Wamel, and Mark B. Van Der Zwaag. Completeness of timed mcrl. *Fundamenta Informaticae*, 361:2002, 2002. [cited at p. 156]

[68] Jan F. Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, October 1992. [cited at p. 10, 21, 30]

[69] Ott User Guide. http://www.cl.cam.ac.uk/ pes20/ott/ott_manual_0.20.3.html. [cited at p. 192]

[70] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing Series. MIT, Cambridge, MA, 1992. [cited at p. 3]

[71] Pieter H. Hartel. Letos - a lightweight execution tool for operational semantics. *Software - Practice and Experience*, 29:1379–1416, December 1999. [cited at p. 156]

[72] Matthew Hennessy. *Algebraic theory of processes*. MIT Press series in the foundations of computing. MIT Press, 1988. [cited at p. 21]

[73] Matthew Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structured Operational Semantics*. Wiley, New York, 1991. [cited at p. 3]

[74] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. [cited at p. 12, 19, 33, 40, 87]

[75] David Hilbert. *The Foundations of Geometry*. Open Court, La Salle, Illinois, tenth edition, 1899. Translated by Leo Unger. [cited at p. 9]

[76] Charles A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978. [cited at p. 121]

[77] Charles A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. [cited at p. 21, 40, 46, 80, 109, 121]

[78] Charles A. R. Hoare, Ian J. Hayes, He Jifeng, Carroll Morgan, A. William Roscoe, Jeff W. Sanders, I. Holm Sorensen, J. Michael Spivey, and Bernard A. Sufrin. Laws of programming. *Communications of the ACM*, 30:672–686, 1987. [cited at p. 19]

[79] Douglas R. Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid (Penguin Philosophy)*. Penguin Books Ltd, new edition, 1990. [cited at p. 8, 9]

[80] Hans Huttel. *Transitions and Trees: An Introduction to Structural Operational Semantics*. Cambridge University Press, 2010. [cited at p. 3]

[81] Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1):1–46, 1996. [cited at p. 186]

[82] Marco Kick. *Coalgebraic Modelling of Timed Processes*. Phd thesis, LFCS, School of Informatics, University of Edinburgh, 2002. [cited at p. 13]

[83] Bartek Klin and Vladimiro Sassone. Structural operational semantics for stochastic process calculi. In Roberto M. Amadio, editor, *Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS 2008)*, volume 4962 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2008. [cited at p. 13]

[84] Matthew R. Lakin. *An executable meta-language for inductive definitions with binders*. PhD thesis, University of Cambridge, 2010. [cited at p. 184, 186]

[85] Matthew R. Lakin and Andrew M. Pitts. A metalanguage for structural operational semantics. In *Symposium on Trends in Functional Programming*, pages 1–16, 2007. [cited at p. 184, 185]

[86] Matthew R. Lakin and Andrew M. Pitts. Resolving inductive definitions with binders in higher-order typed functional programming. In G. Castagna, editor, *18th European Symposium on Programming (ESOP '09)*, volume 5502 of *Lecture Notes in Computer Science*, pages 47–61. Springer, Mar 2009. [cited at p. 184, 186]

[87] Matthew R. Lakin and Andrew M. Pitts. Encoding abstract syntax without fresh names. *Journal of Automated Reasoning*, 2011. To appear. [cited at p. 184, 186]

[88] Kim Guldstrand Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443

of *Lecture Notes in Computer Science*, pages 526–539. Springer-Verlag, 1990. [cited at p. 13, 20, 56]

[89] Eric Madelaine and Didier Vergamini. Finiteness conditions and structural construction of automata for all process algebras. In *Proceedings of the 2nd International Workshop on Computer Aided Verification*, CAV '90, pages 353–363, London, UK, 1991. Springer-Verlag. [cited at p. 34]

[90] Cornelis A. Middelburg. Variable binding operators in transition system specifications. *Journal of Logic and Algebraic Programming*, 47(1):15–45, 2001. [cited at p. 184]

[91] Cornelis A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming (JLAP)*, 55(1-2):1–19, 2003. [cited at p. 184]

[92] Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 118–127, Washington, DC, USA, 2003. IEEE Computer Society. [cited at p. 189]

[93] Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6:749–783, October 2005. [cited at p. 184, 188, 189]

[94] Robin Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977. [cited at p. 11]

[95] Robin Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984. [cited at p. 19]

[96] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. [cited at p. 6, 12, 19, 21, 24, 39, 46, 61, 72, 84, 99, 109, 111, 114, 116, 121, 138, 156, 240]

[97] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i. *Information and Computation*, 100, 1989. [cited at p. 7, 156]

[98] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Information and Computation*, 100:41–77, September 1992. [cited at p. 16, 156, 168, 171]

[99] Robin Milner, Mads Tofte, Robert Harper, and David Macqueen. *The Definition of Standard ML - Revised*. The MIT Press, 1997. [cited at p. 7]

[100] Faron Moller. The importance of the left merge operator in process algebras. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP '90)*, volume 443 of *Lecture Notes in Computer Science*, pages 752–764. Springer-Verlag, 1990. [cited at p. 111]

[101] Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS'90)*, pages 142–153. IEEE Computer Society, 1990. [cited at p. 111]

[102] Peter D. Mosses, Mohammad Reza Mousavi, and Michel A. Reniers. Robustness of equations under operational extensions. Technical Report RR1192, INRIA, 1990. [cited at p. 55, 239]

[103] Peter D. Mosses, Mohammad Reza Mousavi, and Michel A. Reniers. Robustness of equations under operational extensions. In *Proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS'10)*, volume 41 of *Electronic Proceedings in Theoretical Computer Science*, pages 106–120, 2010. [cited at p. 54, 55, 239]

[104] Mohammad Reza Mousavi. Towards sos meta-theory for language-based security. *Electronic Notes in Theoretical Computer Science*, 162:267–271, 2006. [cited at p. 13]

[105] Mohammad Reza Mousavi, Iain Phillips, Michel A. Reniers, and Irek Ulidowski. Semantics and expressiveness of ordered sos. *Information and Computation*, 207(2):85–119, 2009. [cited at p. 13]

[106] Mohammad Reza Mousavi, Iain C. C. Phillips, Michel A. Reniers, and Irek Ulidowski. The meaning of ordered sos. In S. Arun-Kumar and N. Garg, editors, *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 334–345, Kolkata, India, 2006. Springer-Verlag. [cited at p. 13]

[107] Mohammad Reza Mousavi and Michel A. Reniers. Orthogonal extensions in structural operational semantics. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1214–1225. Springer-Verlag, 2005. [cited at p. 24, 54, 239]

[108] Mohammad Reza Mousavi and Michel A. Reniers. Prototyping sos meta-theory in maude. *Electronic Notes in Theoretical Computer Science*, 156:135–

150, May 2006. [cited at p. 156]

[109] Mohammad Reza Mousavi, Michel A. Reniers, and Jan F. Groote. Sos formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3):238–272, 2007. [cited at p. 10, 11, 13, 21, 67, 69, 109, 112, 124, 156, 157, 158, 193, 194, 242]

[110] MohammadReza Mousavi, Michel A. Reniers, and Jan F. Groote. A syntactic commutativity format for SOS. *Information Processing Letters*, 93:217–223, March 2005. [cited at p. 11, 57, 68, 110, 126, 239]

[111] Gopalan Nadathur and Dale Miller. An overview of lambda prolog. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP-SLP 1988)*, pages 810–827. MIT Press, 1988. [cited at p. 189]

[112] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994. [cited at p. 134]

[113] Hanne R. Nielson and Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley Series in Data Communications and Networking for Computer Progammers. wiley, chichester, 1992. [cited at p. 3]

[114] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. [cited at p. 192]

[115] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag. [cited at p. 19, 24, 72, 109, 116]

[116] Giuseppe Peano. *Arithmetices principia, nova methodo exposita*, 1899. [cited at p. 8]

[117] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:2003, 2002. [cited at p. 186, 187]

[118] Gordon D. Plotkin. Lcf considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977. [cited at p. 11]

[119] Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981. [cited at p. 4, 67, 109, 155]

[120] Gordon D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60:60–61, 1981. [cited at p. 6]

[121] Gordon D. Plotkin. An operational semantics for csp. In Dines Bjørner, editor, *Formal Description of Programming Concepts – II*, pages 199–225, Amsterdam, 1983. North-Holland. [cited at p. 7]

[122] The HOL project. http://hol.sourceforge.net/. [cited at p. 192]

[123] Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990. [cited at p. 115]

[124] Teodor C. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, XIII:445–463, 1990. [cited at p. 165]

[125] Arend Rensink. Bisimilarity of open terms. *Information and Computation*, 156(1-2):345–385, 2000. [cited at p. 20, 54, 56, 238]

[126] Jan J. M. M. Rutten. Deriving denotational models for bisimulation from structured operational semantics. In *Proceedings of the IFIP Working Group 2.2/2.3 Working Conference*, pages 155–177. North-Holland, The Netherlands, 1990. [cited at p. 11]

[127] Jan J. M. M. Rutten. Processes as terms: Non-well-founded models for bisimulation. *Mathematical Structures in Computer Science (MSCS)*, 2(3):257–275, 1992. [cited at p. 11]

[128] Jan J. M. M. Rutten and Daniele Turi. Initial algebra and final coalgebra semantics for concurrency. In *Proceedings of REX School/Symposium'1993*, volume 803 of *Lecture Notes in Computer Science*, pages 530–582. Springer-Verlag, 1994. [cited at p. 11]

[129] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996. [cited at p. 166]

[130] Davide Sangiorgi. pi-calculus, internal mobility, and agent-passing calculi. In *Selected papers from the 6th international joint conference on Theory and practice of software development*, TAPSOFT '95, pages 235–274, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V. [cited at p. 194, 242]

[131] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996. [cited at p. 156, 180, 182, 194, 241, 242]

[132] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge Univ. Press, 2001. [cited at p. 16, 137, 156, 168, 171, 175, 176, 179, 180, 181, 182, 194, 208, 241, 242, 243]

[133] David A. Schmidt. *The Structure of Typed Programming Languages*. Foundations of Computing. MIT, Cambridge, MA, 1994. [cited at p. 3]

[134] Wolfram Schwabhäuser, Wanda Szmielew, and Alfred Tarski. *Metamathematische Methoden in der Geometrie*. Springer, Berlin, 1983. [cited at p. 9]

[135] Peter Sewell, Francesco Z. Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strniša. Ott: effective tool support for the working semanticist. *ACM SIGPLAN Notices*, 42:1–12, October 2007. [cited at p. 184, 190, 191, 192]

[136] Mark R. Shinwell. *The Fresh Approach: functional programming with names and binders*. PhD thesis, University of Cambridge, 2005. [cited at p. 186]

[137] Mark R Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*, pages 263–274. ACM Press, 2003. [cited at p. 186]

[138] Alex K. Simpson. Sequent calculi for process verification: Hennessy-milner logic for an arbitrary gsos. *Journal of Logic and Algebraic Programming*, 60-61:287–322, 2004. [cited at p. 12]

[139] Allen Stoughton. *Fully abstract models of programming languages*. Pitman Publishing, Inc., Marshfield, MA, USA, 1988. [cited at p. 11]

[140] Bent Thomsen. A theory of higher order communicating systems. *Information and Computation*, 116(1):38–57, 1995. [cited at p. 166]

[141] Simone Tini. Rule formats for non interference. In *Proceedings of the 12th European Symposium on Programming (ESOP 2003)*, volume 2618 of *Lecture Notes in Computer Science*, pages 129–143, 2003. [cited at p. 13]

[142] Simone Tini. Rule formats for compositional non-interference properties. *Journal of Logic and Algebraic Programming*, 60-61:353–400, 2004. [cited at p. 13]

[143] Irek Ulidowski and Iain C. C. Phillips. Ordered sos process languages for branching and eager bisimulations. *Information and Computation*, 178(1):180–213, 2002. [cited at p. 13]

[144] Irek Ulidowski and Shoji Yuen. Process languages with discrete relative time based on the ordered sos format and rooted eager bisimulation. *Journal of Logic and Algebraic Programming*, 60-61:401–460, 2004. [cited at p. 13]

[145] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004. [cited at p. 156, 157, 159, 161, 193]

[146] Rob J. van Glabbeek. The linear time-branching time spectrum i - the semantics of concrete, sequential processes. In *Handbook of Process Algebra, chapter 1*, pages 3–99. Elsevier, 1999. [cited at p. 39, 61]

[147] Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:229–258, 2004. [cited at p. 164, 165]

[148] Jan van Heijenoort, editor. *From Frege to Gödel: A sourcebook in mathematical logic, 1879 – 1931*. Harvard University Press, 1967. [cited at p. 8]

[149] Muck van Weerdenburg. Automating soundness proofs. In *Proceedings of the 5th Workshop on Structural Operational Semantics (SOS 2008)*, volume 229 of *Electronic Notes in Theoretical Computer Science*, pages 107–118, 2009. [cited at p. 20, 38, 56, 57]

[150] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In *Proceedings of CONCUR '94: Concurrency Theory, 5th International Conference*, volume 836 of *Lecture Notes in Computer Science*, pages 433–448, London, UK, 1994. Springer-Verlag. [cited at p. 10, 238]

[151] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal on Computing*, 2(2):274–302, 1995. [cited at p. 92]

[152] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundation of Computing Series. MIT, Cambridge, MA, 1993. [cited at p. 3]

[153] Axelle Ziegler, Dale Miller, and Dale Palamidessi. A congruence format for name-passing calculi. In *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS'05)*, volume 156 of *Electronic Notes in Theoretical Computer Science*, pages 169–189, Lisbon, Portugal, 2005. Elsevier Science B.V. [cited at p. 194, 243]

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