Concurrent Kleene Algebra

Completeness and Decidability

Tobias Kappé

University College London

Department of Computer Science

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science of University College London, May 2020.

Declaration

I, Tobias Wilhelmus Josephus Kappé, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Concurrent Kleene Algebra (CKA) offers a set of axioms to reason about equivalence of concurrent programs, such that equivalent programs must have the same interpretation in *any* program semantics that respects the axioms of CKA. It builds on the well-known formalism of Kleene Algebra, which offers the same benefits for sequential programs. CKA is *complete*, that is, any valid equivalence can be proved from the axioms. Moreover, equivalence of programs according to CKA can be verified mechanically, i.e., equivalence is *decidable*. Crucial to the latter is the fact that programs can be represented as abstract machines, which admit equivalence checking.

In this thesis, we investigate techniques to augment the reasoning power of CKA with additional truths particular to the program semantics at hand. Building on similar results about Kleene Algebra, we will show that for a large class of extensions, decidability and completeness can be recovered. In particular, our techniques will allow us to incorporate reasoning about *interleaving*, i.e., the partially sequential execution of concurrent programs, as well as *control flow*, such as the conditional branching and looping structures found in most programming languages.

In the second half of this thesis, we will develop our own abstract machine model to represent programs modelled by Concurrent Kleene Algebra, and show that any such program can be modelled by a machine, and vice versa. We will then argue that equivalence of these automata is mechanisable, and that the correspondence between expressions and automata can be extended further to incorporate a more general form of concurrent program composition.

Funding Work on this thesis was supported by the ERC Starting Grant ProFoundNet (679127).

Impact Statement

Outside Academia: Verification of programs is of central importance to the safety of automated systems in daily life. Since the emergence of concurrent or distributed systems and algorithms, this task has become more complex. Because program code is always subject to change, it is important to guarantee that changes do not violate the safety properties of code that was already proved correct. As it happens, verification of program equivalence is undecidable in general. This thesis looks at techniques to prove that general transformations of programs preserve semantics, algebraically as well as mechanically. Because of the hardness of this problem, the main focus is on proving *completeness*, i.e., showing that all valid transformations can be verified, and *decidability*, i.e., the property that this process can, in principle, be automated. The potential payoff of automated equivalence checking is not to be underestimated. For instance, a continuous integration tool could check whether the new version of code is equivalent to the previous (verified) version, thereby avoiding another iteration of unit tests. Similarly, an integrated development environment may suggest several possibly more readable but still equivalent versions of code to the developer. While the practicality of algorithms to check validity of transformations for the sequential case has been well-established by other works, further study is necessary to find out whether the algorithms proposed for concurrent programs in this thesis are feasible in an industrial setting.

Inside Academia: Proofs of completeness and decidability for various extensions of Kleene Algebra has long been treated along the same lines, by reducing these problems to settings where those properties have been established. This thesis formalises these tactic, and establishes useful meta-theory to combine existing reductions, thereby easing the way for the study of future extensions of Kleene Algebra. We show that these techniques can be used to settle open questions about the axiomatisation of interleaving and control flow in concurrent programs. Our approach can also be viewed as a tool to develop new domain-specific programming languages with strong guarantees about semantic equivalence and verifiability, in the mould of NetKAT, an already established programming language for specifying and reasoning about packet routing. Lastly, we propose a new operational formalism for concurrency, and show that this machine model can be used as a proxy to verify concurrent program equivalence. The results in this thesis have been published in several highly ranked conferences [KBL$^+$17; KBS$^+$18; KBR$^+$19; KBS$^+$20] and in one journal article [KBL$^+$19]; at the time of writing, another article is under submission at a journal [KBL$^+$18].

Acknowledgements

It has been almost ten years since I started my formal education in Computer Science. Along the way, I have been blessed to meet and interact with a great number of amazing people.

First and foremost, I am incredibly indebted to Alexandra Silva and Fabio Zanasi, my advisors who afforded me the time and liberty to fall down mathematical rabbit holes. Alexandra, with your cheerful disposition, you created a working atmosphere that is both stimulating and rewarding. Your sage insights and calm tact have made working with you an unreserved pleasure; I sincerely hope that we can continue this collaboration. Fabio, your keen intellect has helped me contextualise the things we worked on, and appreciate them in the broader context of Computer Science. If I ever manage to get half as good as you at writing introductions and conclusions, I will be content.

I would also like to thank my examiners, Robin Hirsch and Georg Struth, for their critical and thorough evaluation of this thesis, and the helpful comments that came with it.

This thesis is based on research that was heavily collaborative and could not have come to fruition if it had not been for my coauthors; a huge thanks to Paul Brunet, Bas Luttik, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. I have also had the privilege to work with people on research outside the scope of this thesis, which provided a welcome change of pace; in particular, I would like to thank Farhad Arbab, Marcello Bonsangue, Simon Docherty, Nate Foster, Gerco van Heerdt, Justin Hsu, Bart Jacobs, Sung-Shik Jongmans, Makoto Kanazawa, Dexter Kozen, Benjamin Lion, Matteo Sammartino, Steffen Smolka, and Carolyn Talcott.

Working conditions in my workplace were always brightened by my officemates, who provided amusing distractions, from our royal family of office plants to an ample stash of candy. My gratitude goes out to Billy, Esteban (who was not formally at UCL, but joined us so often that I might as well include him here), Gerco, Jana, Joshua, Matteo, Paul B., Paul W., Stefan, and Tiago.

Over the course of the last four years, I was fortunate to be able to travel in the margins before or after conferences or summer schools, and I could not have wished for better travel companions. In particular, I have fond memories of the brief holiday up in the Oregon mountains (snow in July!) and the exquisite company I had in Cristina, Jana, and Tao. I will also never forget the food, music, and fun in the handful of days I spent in and around New Orleans with Jana and Stefan.

Back in London, I was blessed to work in an environment full of people always up for a beer in the Marlborough Arms, a Negroni in the Resting Hare, or a (vegan) burger at the Temple of Seitan. My thanks to Anupam, Bas, Benjamin K., Carsten, David, Diana, Fred, Ilya, Joshua, Kareem, Lachlan, Louis, Marguerite, Maria, Paul B., Reuben, Simon C., Simon D., Sonia, and Todd.

It is notoriously hard to find a nice and affordable place to live in London. I therefore count myself extremely lucky to have spent two years living with the best flatmates imaginable: Gerco, Kate and Rorie. A very special thanks goes out to Rorie, who has been an absolute saint for helping me move out of the flat, at a time when the logistics of such a move were highly non-trivial.

Visiting The Netherlands, I was always able to rely on the support of my oldest friends. I am particularly thankful to Adriaan, Frank, and Remco, who kept in touch despite the distance and diligently kept me up to date on the goings-on back home. Every time I went back to Leiden, I was greeted with the smiles of my friends back there. These little visits were always a huge boost to my morale; my thanks to Anna, Bart, Bert, Heleen, Irene H., Irene V., Jeroen, Josse, Murielle, Philipe, Pieter, Simone, Tobias de J., and all of the lovely people at De Leidsche Flesch.

I have two people to thank for starting this PhD in the first place. First of all, Farhad Arbab took me under his wing during my undergraduate and introduced me to the joys of research. Meanwhile, Jurriaan Rot, who had already sparked my interest in coalgebra, first introduced me to Alexandra. Thank you, Farhad and Jurriaan, for helping me during such a pivotal time.

I started and finished this program at the same time as my fellow musketeers, Gerco van Heerdt and Louis Parlant. Guys, thank you so much for all the fun over the past four years, from the good times in our Category Theory reading group to the excellent conversations over lunch.

Many ideas in this thesis were worked out together with Paul Brunet, whose mathematical acumen is second to none. Paul, you are able to keep track of so many layers of abstraction in a complicated proof that it still boggles my mind. Thank you for your patience with my incessant questions, notational shifts, and technical cul-de-sacs. I could not have done this without you.

During the second year of my program, I spent three stimulating months working with Makoto Kanazawa as an intern at the National Institute for Informatics in Tokyo, Japan. To this day, I marvel at Makoto's skill, which included a wide range of counterexamples to my conjectures about context-free languages. Thank you, Makoto, for hosting me and working with me.

After coming back from Japan, I was delighted to work with Jana Wagemaker, who has become a close collaborator. Jana has a sharp mind, amazing intuition, and a knack for posing the right question at the right time. In addition, she has become a close and reliable friend. Jana, thank you for being there; this manuscript and its author would have been worse off if it had not been for you.

Finally, I would like to dedicate this thesis to my parents, Francien and Geert, and my sister, Emma, who together provided both a welcome refuge and a rock-solid base of support. Knowing that you were always there to help me out has made all the difference.

Contents

List of Figures

Chapter 1

Introduction

Information technology pioneer Ted Nelson is said to have quipped *"The good news about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do."* This reflects a critically important aspect of computer science: to automate a task, we have to completely and unambiguously specify the desired behaviour, lest we risk having the program violate its specification. The consequences of the latter in a world pervaded by computing equipment are well-known, ranging from "rapid unplanned disassembly" to loss of human life.

To prevent these adverse effects, programmers need to be provided with the right mathematical and industrial tools that help them prevent mistakes, as well as argue that their code does exactly what it was meant to do. The computer science community has risen to these challenges: from early systems proposed by Floyd [Flo67] and Hoare [Hoa69] to modern program logics such as Concurrent Separation Logic [OHe07; Bro07], program verification is present in undergraduate courses and industry applications alike, and is not likely to lose prominence any time soon.

One way to view the problem of program correctness is through program equivalence [HHH^{+}87]. The underlying thought is that a specification of a program should be equivalent to a program purported to implement that specification, or at the very least, the program should not exhibit any behaviour disallowed by the specification. The added benefit of this shift in perspective is that results about program equivalence can be applied more broadly to programming. For instance, when refactoring a program for readability, it seems reasonable to verify that the resulting code is equivalent to the original, and hence satisfies the same correctness properties. Similarly, we may want to prove that the machine code produced by an optimising compiler is equivalent to the code produced by the programmer, so as to ensure that the intended meaning is not lost in translation.

Of course, verification of program equivalence can be a complex task even in the best circumstances. As a program grows in complexity, so too does the task of confirming whether a given modification preserves its semantics. To allow programmers to quickly adapt their code to new insights, it is therefore desirable to automate equivalence checking as much as possible. Unfortunately, this is where we reach a fundamental limit: since the halting problem is undecidable [Tur37], program equivalence must also be undecidable in general. After all, if we could check program equivalence in general, then we would also be able to solve the halting problem, by fixing the input to the program and checking whether it is equivalent to a program that does not halt.

However, if we abstract from the particular meaning of the statements in the program and instead focus on its structure, i.e., the way that the program arises from these statements, we can still derive meaningful notions of program equivalence that are amenable to mechanisation. More concretely, we can for instance show that a program of the form **if** b **then** e **else** f, which executes e or f depending on whether b is true, is the same as a program of the form **if not** b **then** f **else** e. Practically, this means that we cannot answer the question "are these two programs equivalent?", but we can try to answer the question "does this generic program rearrangement preserve semantics?"

This leaves open the matter of which axioms should be used to reason about program equivalence, and how to mechanise equivalence checking of programs. In this thesis, we study techniques to show (a) that a given set of axioms is *complete*, i.e., that can prove correctness of all valid transformations, and (b) that equivalence under these axioms is *decidable*, i.e., that equivalence between abstract programs under the axioms can be mechanically checked. We focus on programs that permit a form of concurrency known as *fork/join-concurrency*, building on results from Kleene algebra.

Kleene Algebra Regular expressions were proposed by Kleene, as a way to represent the order of events in nerve nets, a model of human neurons [Kle56]. Being a compact way to describe simple patterns in series of symbols or events, regular expressions soon found their way into text editors and programming languages alike. Crucial to this deployment was Kleene's realisation that regular expressions are equivalent to finite automata, known as *Kleene's theorem*. This paved the way for a straightforward algorithm to check acceptance by a regular expression. Among others, Thompson [Tho68] and Brzozowski [Brz64] gave algorithms to convert regular expressions to finite automata; see [Wat93] for a good overview. The other direction of Kleene's theorem, i.e., that every finite automaton corresponds to a regular expression, was further studied by McNaughton and Yamada [MY60] and Backhouse [Bac75]. Silva [Sil10] used the theory of universal coalgebra [Rut00] to transplant Kleene's theorem to new settings, such as Segala systems and Mealy machines.

Regular expressions provided a sufficient but not unique way to describe patterns of events; the problem thus quickly arose: how can one show that two regular expressions denote the same pattern? The most direct answer is to convert the regular expressions in question to finite automata, and to check whether these automata represent the same language. Rabin and Scott [RS59] showed that the latter problem is decidable for deterministic finite automata, and that any finite automaton can be made deterministic. Hopcroft and Karp [HK71] later provided an algorithm to decide in almost linear time [Tar75] whether two finite automata accept the same language. Equivalence checking algorithms were refined over the years, notably by the introduction of the *antichain algorithm* by De Wulf et al. [WDH$^+$06], as well as Bonchi and Pous's algorithm to directly check equivalence of non-deterministic finite automata using *bisimulation up to congruence* [BP13].

Just like expressions in classical algebra, one can reason about regular expressions by manipulating them algebraically. This matter was first studied by Red'ko [Red64a], who proved that no finite set of equations is sufficient to prove all valid equivalences. Soon after, Salomaa [Sal66] provided the first *complete* axiomatisation of regular expressions — i.e., a set of rules that are sufficient to argue equivalence of any two equivalent regular expressions — given by finitely many equations in addition to one rule of inference. This axiomatisation and others were studied by Conway [Con71], and was later generalised for applications to other algebraic systems by Krob [Kro90], Boffa [Bof90] and Kozen [Koz94]. Kozen's axiomatisation of regular expressions is used most widely today, and algebras satisfying these axioms are called *Kleene algebras.* While it is not apparent at this level of detail, it should be noted that the study of axiomatisations of equivalence of regular expressions benefited greatly from Kleene's theorem relating regular expressions and finite automata.

The axioms of Kleene algebra make sense to reason about program equivalence. For instance, choosing non-deterministically between running the programs e and f is the same as choosing between running f and e — in other words, non-deterministic choice is commutative. Nevertheless, this kind of equivalence is very fine-grained, and does not incorporate equalities particular to the program under study. Cohen proposed to study reasoning *under hypotheses* [Coh94], i.e., where in addition to the Kleene algebra axioms, one can use one or more premises that reflect properties of the program at hand. Equivalence under hypotheses is powerful enough to encode general program equivalence, and hence once more undecidable [Koz96]. However, Cohen showed that for certain formats of hypotheses, equivalence remains decidable [Coh94]. This was later studied in more detail by Kozen [Koz02], and extended to more general sets of hypotheses by Kozen and Mamouras [KM14; Mam15]. Most recently, Doumane and collaborators [DKP$^+$19] further extended the sets of hypotheses that permit decidable equivalence, and analysed their complexity.

One particularly useful type of hypotheses allowed reasoning about assertions, and thereby encode control flow structures such as **if** b **then** e **else** f and **while** b **do** e; these *tests* were proposed by Kozen [Koz96]. It was shown that adding the axioms of Boolean algebra is necessary and sufficient to reason about equivalence in a model of *guarded strings* [KS96], and that equivalence in Kleene algebra with tests is just as hard as equivalence in Kleene algebra proper [CKS96]. Kleene algebra with tests then spawned a rich body of work on program equivalence, including applications to program schematology [Koz97; AK01], compiler verification [KP00] and cache control [BK02].

One limitation of Kleene algebra that could not be fixed directly by hypotheses is the lack of primitives for concurrent composition. While interleaving of two concurrent programs could be simulated by including the "shuffle" operator in regular expressions, this kind of concurrency does not reflect the possibility that, especially in multi-processor systems, actions may take place *truly concurrently*, i.e., without any temporal or causal ordering between them. A new model of events to reflect this type of concurrency was proposed by Grabowski [Gra81], and independently by Pratt [Pra82], later studied extensively by Gischer [Gis88]. This model, called *pomsets* by the latter authors, can be thought of as a generalisation of the strictly ordered series of events described by regular expressions to allow partial ordering among events. Pomsets were connected to the shuffle operator by Tschantz [Tsc94], who showed that the equational theory of pomsets coincides with that of languages under concatenation and shuffle — in other words, both sides of any universally valid equation of languages involving these operators correspond to the same pomset.

The development of pomsets as a model of concurrent events led Lodaya and Weil [LW00] to study an extension of regular expressions known as *series-parallel rational expressions*, with a model in terms of pomsets. In op. cit., an operational model to accept these *pomset languages* is also put forward, along with a two-way correspondence between (a fragment of) this operational model and series-parallel rational expressions. Lodaya, Ranganayakulu and Rangarajan proved a similar Kleene theorem relating a fragment of series-parallel rational expressions to 1-safe Petri nets [LRR03]. Hoare and collaborators [HMS$^+$09] proposed to study *concurrent Kleene algebra*, i.e., a concurrent extension of Kleene algebra. Meanwhile, Prisacariu proposed an alternative model called *synchronous Kleene algebra*, which differed from concurrent Kleene algebra in that it allowed only single actions to take place concurrently. Laurence and Struth [LS14] followed up on [HMS$^+$09] by showing that part of the axioms of concurrent Kleene algebra were sound and complete for a straightforward semantics of series-parallel rational expressions in terms of pomsets. Jipsen and Moshier [JM16] proposed another operational model for pomset languages, and made the first steps to extend concurrent Kleene algebra with tests in order to reason about control flow.

Process algebra No discussion of concurrent program equivalence would be complete without mentioning process algebra. Collectively, process algebra refers to the study of calculi such as CSP [Hoa78], CCS [Mil80] and ACP [BK87]. While we cannot do justice to the rich and diverse body of work around process algebra, we shall venture to provide a brief comparison with Kleene algebra here. In the remainder of this thesis, we limit our scope to concurrent Kleene algebra.

Process algebra, like Kleene algebra, can be used to show that two programs are equivalent. The main difference with Kleene algebra is in granularity: whereas equivalence in Kleene algebra typically means that both programs can exhibit the same patterns of events in the same order, equivalence in process algebra is usually defined using some form of *bisimulation*, where two processes are bisimilar if one process can mimic the computational steps of the other, and vice versa. Hence, Kleene algebra can typically be used to prove more programs equivalent, at the cost of abstracting away from behavioural differences which may be relevant to the interpretation. Checking of bisimulation also lends itself more readily to mechanisation; almost every definition of bisimulation readily be checked by means of an algorithm that arises straightforwardly from said definition.

Algebraic equivalence under bisimulation has been studied extensively [AF06; AI07]. A subtle but significant difference with Kleene algebra with regard to axiomatisation is that the syntax of process calculi is usually set up differently from Kleene algebra, in that primitive actions are not processes on their own but rather operators of the form $\mathtt{a}.-$, which take a process and prefix it with an action $\mathtt{a}$. This has ramifications for the power of algebraic reasoning: whereas equivalences in Kleene algebra are “scale-free” in that any primitive actions can be substituted with other expressions to obtain an equally valid equivalence, the same does not hold true for process algebra.

Lastly, process calculi have historically included concurrent (interleaving) composition from the start, while (as demonstrated above) this inclusion is relatively new in the realm of Kleene algebra.

Roadmap We will start by laying down some preliminary notions and notation in Chapter 2. Next, we formally define pomsets and the axioms of bi-Kleene algebra in Chapter 3. The first technical part of this thesis is focused on algebraic reasoning, organised as follows:

- In Chapter 4, we will start by showing how bi-Kleene algebra can be extended with hypotheses, building on [DKP+19], and discussing techniques to argue decidability as well as completeness (Lemmas 4.27, 4.32, 4.41 and 4.49) w.r.t. a model in terms of pomset languages (Theorem 4.14).
- In Chapter 5, we encode the exchange law in hypotheses, and argue that these are complete w.r.t. a pomset language model (Corollary 5.38), settling a conjecture by Hoare et al. [HSM+16]. We also show how the exchange law can be separated from other hypotheses (Theorem 5.52).

$$\begin{tikzcd}
{[\mathrm{KBS}^{+}18]} \arrow[dr] & {[\mathrm{KBS}^{+}20]} \arrow[dl] \arrow[d] \arrow[dr] & {[\mathrm{KBR}^{+}19]} \arrow[d] & {[\mathrm{KBL}^{+}17]} \arrow[d] & {[\mathrm{KBL}^{+}18]} \arrow[dl] \arrow[d] & {[\mathrm{KBL}^{+}19]} \arrow[dl] \arrow[d] \\
\text{Chapter 4} & \text{Chapter 5} & \text{Chapter 6} & \text{Chapter 7} & \text{Chapter 8} & \text{Chapter 9}
\end{tikzcd}$$

Figure 1.1: Redistribution of material from published and submitted papers among chapters.

- In Chapter 6, we extend concurrent Kleene algebra with a mechanism to reason about control flow. We show that an extension along the same lines as Kleene algebra with tests [JM16] is unsuitable for reasoning about programs (Fact 6.6), and argue that a slight weakening of the axioms appropriate to concurrency can be used to get around this problem (Corollary 6.38).

We conclude our study of algebraic reasoning by listing directions for further work. In the second half of this thesis, we focus on extending Kleene's theorem to the concurrent setting.

- In Chapter 7, we propose *pomset automata*, an operational model of pomset languages. We show how Antimirov's construction [Ant96] can convert a series-rational expression into pomset automaton representing the same language (Theorem 7.35), and how a well-defined class of pomset automata can also be represented by series-rational expressions (Theorem 7.41).
- In Chapter 8, we argue that equivalence of the *fork-acyclic* fragment of pomset automata is decidable (Corollary 8.54), using a method inspired by the work of Laurence and Struth [LS14]. We also prove that equivalence of general pomset automata is undecidable (Corollary 8.10), justifying the restriction to fork-acyclic pomset automata.
- In Chapter 9, we discuss an extension of series-rational expressions known as *series-parallel rational expressions*, and extend the earlier Kleene theorem to include these. In particular, we isolate a strictly more expressive fragment of pomset automata called *well-nested pomset automata* that corresponds exactly to these expressions (Corollary 9.36).

Finally, we conclude this thesis by listing open questions regarding pomset automata.

Relation to co-authored works This thesis is based on joint work with (in alphabetical order) Paul Brunet, Bas Luttik, Jurriaan Rot, Alexandra Silva, Jana Wagemaker and Fabio Zanasi, although any errors in the present manuscript are mine. The material appearing in the technical chapters is based on six papers. For the sake of narrative, the content of these papers has been redistributed among six thematically consistent chapters.

The backbone of the first half is formed by [KBS$^+$20]. In particular, most of the results in Chapter 4 first appeared in op. cit., including the application of hypotheses, reductions, and factorisations to pomset languages, although the results about factorisations have been generalised substantially. The reduction presented in Chapter 5 is based on [KBS$^+$18], but the mathematics have been polished and reworked to fit in the framework set by the previous chapter. Most of the results about factorisation in this chapter first appeared in [KBS$^+$20], with the exception of the dualisation of results about left-simple hypotheses to right-simple hypotheses, which is new. The contents of Chapter 6 mostly originate from [KBR$^+$19] and have been adapted to fit earlier discourse; the main results presented there constitute the last section of [KBS$^+$20].

In the second half, the results that appear in Chapter 7 were first published in [KBL$^+$17] and updated and polished substantially for [KBL$^+$18]. The results about decidability in [KBL$^+$18] form the basis of Chapter 8; on the other hand, results about undecidability are adapted from [KBL$^+$19]. Chapter 9 is based mostly on [KBL$^+$19], although the presentation has been adapted to fit with the other chapters. In Chapters 7 and 9, the formulation of the automata-to-expressions construction in terms of the solution to a (series-rational or series-parallel rational) system is new. Everywhere except Chapter 8 the narrative has been thoroughly rewritten based on the original work.

Figure 1.1 contains a diagram summarising the correspondence between chapters and papers.

How to read this thesis The halves of this thesis are largely self-contained, save for the odd cross-reference. Definitions and notations used in both parts are discussed in Chapters 2 and 3; in particular, the solutions to (series-rational) systems will make an appearance in both halves.

Most of the work to prove the theorems in the technical chapters goes into finding the right construction; validating the correctness of the construction is often routine. For this reason, the main text will place emphasis on how the relevant construction is derived from the constraints at hand, and strive to provide intuition as to why this construction might work. Most proofs are then outlined by going through a series of lemmas, which naturally lead to the conclusion presented in the form of a theorem. The proofs of these lemmas appear in the appendices so as not to interrupt the narrative. The interested reader is invited to move between the main text and the proofs to get a full explanation of all the details. For convenience, the headings provide hyperlinks to jump back and forth — i.e., the first appearance of “Lemma X” links to the proof in the appendix, and the second appearance links back to the original statement. Some of the more pivotal proofs are included in the main text for the benefit of the reader’s understanding; with the exception of a handful, these are short and rely on earlier stated and explained lemmas.

Chapter 2

Preliminaries

For the sake of self-containment, we start by specifying some mathematical notation. Readers familiar with the mathematical vernacular may want to skip ahead to Chapter 3.

Sets Objects can be gathered in *sets*; we use brackets $\{-\}$ to denote the contents of a set. For instance, the set of natural numbers $\mathbb{N}$ is given by $\{0, 1, 2, \dots\}$. The order of elements does not matter, nor does any repetition; hence $\{1, 0, 0\}$ is the same set as $\{0, 1\}$. When an object a appears in a set A, we say that *a is an element of A*, denoted $a \in A$; when a is *not* an element of A, we write $a \notin A$. When A and B are sets such that all elements of A are also elements of B, we say that A is a *subset* of B, denoted $A \subseteq B$. Sets A and B are equal when they are subsets of one another, i.e., when $A \subseteq B$ and $B \subseteq A$. We can also specify a set by taking another set and specifying a predicate on elements of the latter. For instance, we can specify the set of even numbers by $\{n \in \mathbb{N} : n \text{ is even}\}$, or more precisely $\{n \in \mathbb{N} : \text{there exists } k \in \mathbb{N} \text{ such that } 2k = n\}$.

Another way of building a set is by saying that it is the smallest set satisfying some inference rules. For instance, we can build the set of even numbers by saying A is the smallest set satisfying

$$\frac{}{0 \in A} \qquad\qquad \frac{n \in A}{n + 2 \in A}$$

In these rules, the premises appear above the line, and the consequences appear below the line. The first rule should thus be read as "0 is always an element of A", and the second line should be read as "if n is an element of A, then so is $n + 2$". Formally, these inference rules should induce a monotone operator on sets, which (by the Knaster-Tarski fixpoint theorem) admits a unique fixpoint. This condition can usually be satisfied by avoiding negation when writing the rules.

A useful consequence of specifying a set using inference rules is that, to show that this set A is contained in a set B, all one has to do is show that B satisfies the inference rules that build A. After all, A is the *smallest* set satisfying those rules, so if B satisfies the same rules, then A must necessarily be a subset of B. We will make repeated use of this principle throughout this thesis.

We will repeatedly use inference rules as an inductive handle in proofs. For instance, if A is defined using inference rules and we want to prove that all members of A satisfy a certain property, should show that if all elements of A that appear in the premise of an inference rule satisfy this property, then so do the members of A that appear in the consequence of the rule.

Set composition The *union* of two sets A and B, denoted $A \cup B$, is the smallest set containing the elements of A and B; similarly, the *intersection* of two sets, denoted $A \cap B$, is the largest set containing elements that are both in A and B. We write $A \setminus B$ for the *difference* of A and B, which is the set of elements that appear in A but not in B. The *empty set*, i.e., the unique set having no elements at all, is denoted $\cap$. For instance, if A consists of the even numbers $\{0, 2, 4, \dots\}$ and B consists of the odd numbers $\{1, 3, 5, \dots\}$, then $A \cup B = \mathbb{N}$, $A \cap B = \emptyset$ and $A \setminus B = A$. When two sets A and B have no elements in common, i.e., when $A \cap B = \emptyset$, we say that they are *disjoint*.

The set of subsets of a set A, also called the *powerset* of A, is denoted 2^A. For instance, we have that $\{0, 1\} \in 2^{\mathbb{N}}$. More generally, it holds that $A \in 2^B$ if and only if $A \subseteq B$.

A *tuple* is a finite and ordered list of objects. We will use angular brackets $\langle - \rangle$ to denote tuples, such as $\langle 0, 1 \rangle$ for the two-element tuple containing 0 and 1. Order and repetition matters — i.e., $\langle 0, 1 \rangle$ is *not* the same as $\langle 1, 0 \rangle$, nor $\langle 0, 0, 1 \rangle$. When A and B are sets, we write $A \times B$ for their *Cartesian product*, which contains all tuples $\langle a, b \rangle$ where $a \in A$ and $b \in B$. We write A^n for the n-fold Cartesian product of A, e.g., $A^2 = A \times A$. We flatten tuples, i.e., $\langle \langle 0, 1 \rangle, 2 \rangle$ is the same as $\langle 0, \langle 1, 2 \rangle \rangle$ and $\langle 0, 1, 2 \rangle$; hence, we think of $A \times (B \times C)$ as equal to $(A \times B) \times C$, writing $A \times B \times C$.

Relations A *relation* between sets A and B is a subset of $A \times B$. We commonly use infix notation to denote membership of a relation, i.e., when $R \subseteq A \times B$ is a relation between A and B, we write $a \mathrel{R} b$ to denote $\langle a, b \rangle \in R$; we also say that *$a$ is related to b* by R. Note that, since relations are sets, we can specify them using the same inference rule format that we discussed for sets.

When R is a relation between A and itself, we say that R is a relation *on* A. Let R be a relation on A. We say that R is *reflexive* if for all $a \in A$ it holds that $a \mathrel{R} a$; it is *irreflexive* if for all $a \in A$ it does *not* hold that $a \mathrel{R} a'$. Furthermore, R is *symmetric* if for all $a, a' \in A$ with $a \mathrel{R} a'$ it holds that $a' \mathrel{R} a$; it is *transitive* if for all $a, a', a'' \in A$ with $a \mathrel{R} a'$ and $a' \mathrel{R} a''$ it holds that $a \mathrel{R} a''$. Lastly, R is *antisymmetric* if for all $a, a' \in A$ such that $a \mathrel{R} a'$ and $a' \in R\ a$ it holds that $a = a'$.

A relation R is a *preorder* if it is reflexive and transitive and a *partial order* if it is a preorder that is antisymmetric. A *partially ordered set (poset)* is a pair $\langle A, R \rangle$, where A is a set and R is a partial order on A. R is a *strict order* if it is irreflexive and transitive. Any preorder R gives rise to a strict order R' by saying that for $a, a' \in A$ we have that $a \mathrel{R'} a'$ when $a \mathrel{R} a'$ while $a' \mathrel{R} a$ does not hold. We often denote a partial order with the ordering symbol $\leq$. When we do this, we shall use $<$ to denote the largest strict order contained in $\leq$, i.e., $a < b$ if and only if $a \neq b$ and $a \leq b$.

A relation R is an *equivalence* if it is reflexive, symmetric and transitive. When A is a set and R is an equivalence on A, we write $[a]_R$ for the *equivalence class* of $a \in A$, which is the set $\{a' \in A \; : \; a \mathrel{R} a'\}$. We shall drop the subscript when the equivalence is clear from context.

When A, B and C are sets and $R \subseteq A \times B$ and $R' \subseteq B \times C$ are relations, we write $R \circ R'$ for the *relational composition* of R and R', which is the smallest relation such that if $a \mathrel{R} b$ and $b \mathrel{R'} c$, then $a \mathrel{(R \circ R')} c$. Furthermore, when R is a relation on a set A, we write R^* for the *reflexive and transitive closure* of R, which is the least reflexive and transitive relation on A that contains R.

Functions A *function* f from A to B, denoted $f : A \to B$ is a relation between A and B such that for every $a \in A$ there exists exactly one $b \in B$ such that a is related to b by f. Given $a \in A$, we denote the associated element of b by $f(a)$. Note that f is completely specified by choosing for every $a \in A$ the associated $f(a) \in B$. We say that f is a *bijection* if for every $b \in B$, there exists exactly one $a \in A$ with $f(a) = b$. When $g : B \to C$ is a functions, we write $g \circ f$ the *composition* of g and f, i.e., where $(g \circ f)(a) = g(f(a))$. We use B^A to denote the set of functions from A to B.[1]

When $f : A \to B$ is a function, call A the *domain* of f, and B the *codomain* of f. If the domain of f is a Cartesian product, i.e., $A = A_1 \times \cdots \times A_n$, we drop the tuple brackets when referring to the function's values, i.e., we write $f(a_1, \ldots, a_n)$ instead of $f(\langle a_1, \ldots, a_n \rangle)$. Some functions are denoted differently, in which case we will use the symbol $-$ as a placeholder for the input. For instance, we will often use $[\![-]\!]$ as a function, and $[\![e]\!]$ when referring to the value of $[\![-]\!]$ for input e.

When f is a function from A to 2^B, we can use inference rules to specify f, similarly to how we can use inference rules to specify a set. Formally, f is the smallest function satisfying those inference rules, or alternatively we can think of them as a system of rules to specify a set $f(a)$ for every $a \in A$, possibly with cross-references to members of $f(a')$ for some $a' \in A$. Either way, the same condition applies: the inference rules should induce a monotone operator on the set of functions from A to 2^B; all such definitions in this thesis tacitly satisfy this restriction.

[1]When we regard 2 as the two-element set $\{0, 1\}$, the notation 2^A can be interpreted to mean either the powerset of A or the set of functions from A to 2; it should be obvious that these sets are in one-to-one correspondence.

When $\langle A, \leq \rangle$ and $\langle A', \leq' \rangle$ are posets and $f : A \to A'$ is a function, we say that f is a *poset morphism* if f preserves the order, that is, if for $a_0, a_1 \in A$ with $a_0 \leq a_1$ it holds that $f(a_0) \leq' f(a_1)$.

A *closure operator (on a set A)* is a function that takes a subset of A and "closes" it, in the sense that it sends each set to the smallest set satisfying some property. Formally, a closure operator is a function $f : 2^A \to 2^A$ such that, for all $B, C \subseteq A$, it holds that $B \subseteq f(C)$ if and only if $f(B) \subseteq f(C)$. In particular this means that, for all $B \subseteq A$, it holds that $B \subseteq f(B)$. Furthermore, it follows that closure is monotone w.r.t. the inclusion order on 2^A, that is, if $C \subseteq B$, then $f(C) \subseteq f(B)$. The operator $-^*$ discussed above, which associates with each relation on a fixed set the smallest reflexive and transitive relation that contains it, is one example of a closure operator.

Multisets A *multiset* is a collection of objects where order does not matter, but repetition does. We will use double brackets $\{\!|\, - \,|\!\}$ to explicitly denote finite multisets, such as $\{\!|0, 1, 2, 2|\!\}$. Note the duplicate appearance of 2 in this multiset, which distinguishes it from $\{\!|0, 1, 2|\!\}$. All multisets in this thesis will be finite; hence, we can formally think of a multiset over a set A as a function ϕ from A to $\mathbb{N}$, where $\phi(a)$ specifies how often the element $a \in A$ appears in ϕ, if at all.

When ϕ is a multiset, we use $|\phi|$ to denote the number of elements in ϕ, including multiplicity. More formally, if ϕ is a multiset over A, then $|\phi|$ is defined to be the sum of $\phi(a)$ for all $a \in A$. For instance, if $\phi = \{\!|0, 1, 2, 2|\!\}$, then $|\phi| = 4$, since $\phi(0) + \phi(1) + \phi(2) = 4$.

We use $\mathbb{M}(A)$ to denote $\mathbb{N}^A$, i.e., the set of all multisets containing elements of A. When ϕ and ψ are multisets over A, we write $\phi \sqcup \psi$ for their disjoint union (or sum), which is the multiset where $(\phi \sqcup \psi)(a) = \phi(a) + \psi(a)$. For instance, when $\phi = \{\!|0, 1|\!\}$ and $\psi = \{\!|1, 2|\!\}$, then $\phi \sqcup \psi = \{\!|0, 1, 1, 2|\!\}$. We also use $\binom{A}{n}$ to denote the set of n-ary multisets over A, i.e., the set of $\phi \in \mathbb{M}(A)$ such that the sum of all $\phi(a)$ for $a \in A$ is n. For instance, $\{\!|0, 1|\!\}$ appears in $\binom{A}{2}$, but $\{\!|0, 1, 1|\!\}$ does not.

Words An *alphabet* is a (usually, but not always, finite) set of symbols. A *word* over some alphabet Σ is a finite sequence of symbols from Σ; for instance, if $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, then $\mathtt{aba}$ is a word over Σ. Words can be *concatenated*, that is, we can append one word to another, by the concatenation operator $\cdot$; for instance, $\mathtt{ab} \cdot \mathtt{ba} = \mathtt{abba}$. The empty word (i.e., without any letters) is denoted 1.[2] When $n \in \mathbb{N}$ and $\mathtt{a} \in \Sigma$, we write $\mathtt{a}^n$ for the n-fold concatenation of $\mathtt{a}$, i.e., $\mathtt{a}^0 = 1$ and $\mathtt{a}^{n+1} = \mathtt{a} \cdot \mathtt{a}^n$. The empty word is neutral w.r.t. concatenation, i.e., if w is a word, then $1 \cdot w = w = w \cdot 1$. A set of words is called a *language*; the language of all words over an alphabet Σ is denoted Σ^*.

[2]Most authors use either the symbol ϵ or λ to denote the empty word; we choose 1 to conveniently identify the empty word and the empty *pomset*, defined in the next chapter.

Expressions We will often build *expressions* using symbols that represent functions with one or more inputs, and constants, over a fixed alphabet. Formally, these expressions are built inductively, using inference rules. For instance, we could describe the set of numerical expressions over some fixed set of variables V as the smallest set E satisfying the following inference rules:

$$\frac{n \in \mathbb{N}}{n \in E} \qquad \frac{v \in V}{v \in E} \qquad \frac{e, f \in E}{e + f \in E} \qquad \frac{e, f \in E}{e \times f \in E} \qquad \frac{e \in E}{-e \in E}$$

To ease definitions of syntax, we may use a *(Backus-Naur form) grammar*. For instance, the set of expressions defined above could equivalently be said to be generated by the grammar

$$e, f ::= n \in \mathbb{N} \;\mid\; v \in V \;\mid\; e + f \;\mid\; e \times f \;\mid\; -e$$

It is important to remember that elements of E are purely syntactic objects; they do not take on any meaning until we provide a way to evaluate them. We shall use parentheses to disambiguate the binding of expressions when necessary, writing for instance $(1 + v) \times 2 \in E$ to denote the expression obtained by applying the first two rules to find that $1, v \in E$, applying the third rule to find that $1 + v \in E$, and finally applying the first and last rule to find that $(1 + v) \times 2 \in E$. We may also specify a precedence among operators; for instance, when we say that $\times$ "takes precedence over" (or "binds more tightly than") $+$, the expression $1 + 2 \times 3$ is meant to denote $1 + (2 \times 3)$.

A relation R on a set of expressions is a *congruence* if it is an equivalence compatible with the operators. For instance, for the set of expressions E defined above, a congruence R on E would be an equivalence such that if $e_1 \mathrel{R} f_1$ and $e_2 \mathrel{R} f_2$, then $e_1 + e_2 \mathrel{R} f_1 + f_2$, and similarly for the other operators. Since such an implication can be encoded using inference rules, we can speak of the *smallest* (or *least*) congruence satisfying some given set of inference rules.

A relation *precongruence* R on a set of expressions is a preorder on those expressions that is compatible with the operators in the same way that a congruence is. For instance, a precongruence on the set of expressions E defined above would be a preorder on E such that (among other things) if $e_1 \mathrel{R} f_1$ and $e_2 \mathrel{R} f_2$, then $e_1 \times e_2 \mathrel{R} f_1 \times f_2$, with similar rules for the other operators. Such a property can again be encoded in inference rules, and hence we can speak of the *least precongruence.*

Automata Let Σ be an alphabet. A *non-deterministic automaton* (*NA*) is a tuple $\langle Q, F, \delta \rangle$, where Q is a set of *states*, and F is a subset of Q whose elements are called *accepting states.* Finally, $\delta : Q \times \Sigma \to 2^Q$ is a function called the *transition function.* When there are finitely many states, the non-deterministic automaton is said to be *finite*; such an automaton is called an *NFA*. All of the non-deterministic automata in this thesis will be finite, and will use a finite alphabet.

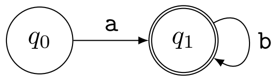


Figure 2.1: An example non-deterministic automaton.

We may graphically depict an NFA as in Figure 2.1. Here, every state is drawn as a (usually circular) shape containing the symbol that state, with doubly circled circles representing the accepting states. The transition function is represented by the arrows between the states, labelled by the alphabet symbol associated with that transition. Thus, in the example drawing, we see that q_0 and q_1 are states, with q_1 accepting, and that $q_1 \in \delta(q_0, \mathtt{a})$ as well as $q_1 \in \delta(q_1, \mathtt{b})$.

Given an NA $A = \langle Q, F, \delta \rangle$, we can define the function $L_A : Q \to \Sigma^*$, which sends every state $q \in Q$ to the *language accepted by* q. Intuitively, this is the set of words that can be read by following the arrows of the transition function starting in q and leading to an accepting state. Formally, L_A is the smallest function satisfying the inference rules

$$\frac{q \in F}{1 \in L_A(q)} \qquad\qquad \frac{q' \in \delta(q, \mathtt{a}) \qquad w \in L_A(q')}{\mathtt{a} \cdot w \in L_A(q)}$$

For instance, one can show that if A is the NFA from Figure 2.1, then $L_A(q_0) = \{\mathtt{a} \cdot \mathtt{b}^n \;:\; n \in \mathbb{N}\}$.

Decidability A *decision problem* poses a well-defined set of inputs, and asks whether those inputs satisfy a certain formally defined property. We say that a decision problem is *decidable* if there exists an algorithm, that is, an unambiguous sequence of steps and tests, that can be used to find out in finitely many steps whether a given input does or does not satisfy the constraints of the problem. What it means for inputs to be well-defined and what qualifies as an algorithm is left unspecified on purpose; these concepts can be formalised, but are outside the scope of this thesis. Suffice to say that some problems, such as the *halting problem* [Tur37], are *undecidable*.

Similarly, we say that a function $f : A \to B$ is *computable* when there exists an algorithm that, given $a \in A$, can compute $f(a)$. Computable functions are important when showing that instances of one decision problem can be converted into instances of another decision problem while preserving the answer, thereby showing that the former problem is decidable when the latter problem is.

Chapter 3

Pomsets and sr-expressions

In this chapter, we discuss two concepts that are foundational to the development in the rest of the thesis: *pomsets* and *series-rational expressions*. It should be stressed that no claim of originality is made to any of the material in this chapter; all definitions and results are adapted from existing literature, chiefly [Gra81; Gis88; LW00; LS14; Ard61; Con71] to fit the purpose of this thesis.

An execution of a program can be seen as a sequence of symbols, each representing an action. For instance, consider an automatic tea dispenser: $\texttt{beep} \cdot \texttt{tea} \cdot \texttt{milk} \cdot \texttt{beep}$ could represent the behaviour where the machine beeps, dispenses tea followed by milk, and then beeps, in that order.

To represent an execution of a program with concurrency, we need to relax this model to allow a partial order on events. For instance, a concurrency-enabled tea dispenser could beep, then *fork* execution into two threads that provide milk and tea *at the same time*, before *joining* the threads to beep again (c.f. Figure 3.1). Note that the events labelled by `milk` and `tea` are ordered after the first occurrence of `beep`, but there is no ordering between them — they are *concurrent*.

The objects most commonly found in the literature to account for such executions are known as *partial words* [Gra81], or equivalently, *partially ordered multisets* (*pomsets*) [Pra82]. In this chapter, we review basic material on pomsets found in the literature [Gra81; Pra82; Gis88].

Defining pomsets requires some care, as the indirection between *events* and their *names* is not

$$\begin{tikzcd}
\texttt{beep} \arrow[r] \arrow[d] & \texttt{tea} \arrow[d] \\
\texttt{milk} \arrow[r] & \texttt{beep}
\end{tikzcd}$$

Figure 3.1: Example ordering of events in a program.

entirely straightforward to write down formally. We start with a basic model of concurrent events in the form of *labelled partially ordered sets*, which we refine momentarily.

Convention 3.1. We fix an alphabet Σ of symbols commonly referred to as *actions.* When using an object S defined using Σ with a different alphabet, Δ, we shall make this explicit, writing $\mathsf{S}(\Delta)$.

Definition 3.2 (Labelled posets)**.** A *labelled poset* is a tuple $\mathbf{u} = \langle S, \leq, \lambda\rangle$ consisting of some *carrier* set S, where $S \subseteq \mathbb{N}$,[1] a partial order $\leq$ on S and a *labelling* function $\lambda : S \to \Sigma$.

When $\mathbf{u}$ is a labelled poset, we write $S_\mathbf{u}$, $\leq_\mathbf{u}$ and $\lambda_\mathbf{u}$ for the carrier, partial order and labelling of $\mathbf{u}$ respectively. The set of labelled posets is denoted LP, and the empty labelled poset is $\mathbf{1}$.

Example 3.3. The execution in Figure 3.1 can be represented by the labelled poset $\mathbf{u}$, where

$$S_\mathbf{u} = \{1,2,3,4\} \qquad 1 <_\mathbf{u} 2 <_\mathbf{u} 4 \qquad 1 <_\mathbf{u} 3 <_\mathbf{u} 4 \qquad \lambda_\mathbf{u} = \left\{ \begin{array}{ll} 1 \mapsto \mathtt{beep}, & 2 \mapsto \mathtt{tea}, \\ 3 \mapsto \mathtt{milk}, & 4 \mapsto \mathtt{beep} \end{array} \right\}$$

Suppose we construct the labelled poset $\mathbf{v}$ just like $\mathbf{u}$ in the example above, but we use 5 instead of 2 for the event labelled with `tea`. Now $\mathbf{v}$ is just as accurate a representation of the execution in Figure 3.1 as $\mathbf{u}$. Hence, the exact carrier of the labelled poset does not matter, but the ordering and labelling of the events do. We formalise the correspondence between $\mathbf{u}$ and $\mathbf{v}$ as follows.

Definition 3.4 (Labelled poset isomorphism)**.** Let $\mathbf{u}$ and $\mathbf{v}$ be labelled posets. A *labelled poset morphism* from $\mathbf{u}$ to $\mathbf{v}$ is a poset morphism from $\langle S_\mathbf{u}, \leq_\mathbf{u}\rangle$ to $\langle S_\mathbf{v}, \leq_\mathbf{v}\rangle$ such that $\lambda_\mathbf{v} \circ h = \lambda_\mathbf{u}$. Moreover, h is a *labelled poset isomorphism* if it is a bijection such that h^{-1} is a poset isomorphism from $\mathbf{v}$ to $\mathbf{u}$. We say that $\mathbf{u}$ *is isomorphic to* $\mathbf{v}$, denoted $\mathbf{u} \cong \mathbf{v}$, if there exists a poset

Example 3.5. The labelled posets $\mathbf{u}$ and $\mathbf{v}$ discussed above are isomorphic, as witnessed by the isomorphism that switches 2 for 5 and leaves the other elements untouched.

Note that $\cong$ is an equivalence. We can then use $\cong$ to abstract from the carrier, as follows.

Definition 3.6 (Pomsets)**.** A *partially ordered multiset*, or *pomset*, is a $\cong$-equivalence class of labelled posets; we write $[\mathbf{u}]$ for the $\cong$-equivalence class of $\mathbf{u} \in \mathsf{LP}$, i.e., $[\mathbf{u}] = \{\mathbf{v} \in \mathsf{LP} \ : \ \mathbf{u} \cong \mathbf{v}\}$.

We write Pom for the set of pomsets. The $\cong$-equivalence class of $\mathbf{1}$ is denoted by 1. We identify $\mathtt{a} \in \Sigma$ with the pomset containing exactly one event, labelled $\mathtt{a}$; such a pomset is called *primitive*.

Convention 3.7. We tacitly assume w.l.o.g. that all (finitely many) pomsets in scope are represented by labelled posets with disjoint carriers. All pomsets in the sequel are finite.

[1] Restricting the carrier to contain natural numbers exclusively makes the collection of labelled posets a proper set.

3.1 Composition

It often makes sense to describe the execution of a program in terms of smaller executions; for instance, we could informally describe an execution by saying *these events were taking place while these other things occurred*, or *all of these things happened, followed by all of these other actions.*

Since pomsets are meant to be models of program execution, it makes sense to compose them similarly, both in parallel (the first option) and in sequence (the second option).

Definition 3.8 (Pomset composition)**.** Let $U, V \in \mathsf{Pom}$ with $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$. We write $U \parallel V$ for the *parallel composition* of U and V, which is the pomset represented by $\mathbf{u} \parallel \mathbf{v}$, where

$$S_{\mathbf{u}\parallel\mathbf{v}} = S_{\mathbf{u}} \cup S_{\mathbf{v}} \qquad \leq_{\mathbf{u}\parallel\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}} \qquad \lambda_{\mathbf{u}\parallel\mathbf{v}}(x) = \begin{cases} \lambda_{\mathbf{u}}(x) & x \in S_{\mathbf{u}} \\ \lambda_{\mathbf{v}}(x) & x \in S_{\mathbf{v}} \end{cases}$$

Similarly, we write $U \cdot V$ for the *sequential composition* of U and V, that is, the pomset represented by the labelled poset $\mathbf{u} \cdot \mathbf{v}$, with the carrier, ordering and labelling function given by

$$S_{\mathbf{u}\cdot\mathbf{v}} = S_{\mathbf{u}\parallel\mathbf{v}} \qquad \leq_{\mathbf{u}\cdot\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}} \cup (S_{\mathbf{u}} \times S_{\mathbf{v}}) \qquad \lambda_{\mathbf{u}\cdot\mathbf{v}} = \lambda_{\mathbf{u}\parallel\mathbf{v}}$$

Example 3.9. Let be $\mathbf{u}$ as in Example 3.3. If we represent the primitive pomset `beep` by the labelled poset $\langle\{1\}, \{\langle 1, 1\rangle\}, \{1 \mapsto \texttt{beep}\}\rangle$ and similarly use 2, 3 and 4 as events to represent `tea`, `milk` and the second occurrence of `beep` respectively, we find that $\mathbf{u}$ is precisely the labelled poset built by $\texttt{beep} \cdot (\texttt{tea} \parallel \texttt{milk}) \cdot \texttt{beep}$. In this labelled poset, the event 1, labelled by `beep`, precedes all of the other events, but the events 2 and 3 (labelled by `tea` and `milk` respectively) are unordered.

Remark 3.10. The operators above yield the same pomset regardless of the labelled posets chosen to represent the operands. Other pomset-related notions will also enjoy this property.

Also, it is easy to see that both operators are associative, and that $\parallel$ is commutative. Furthermore, 1 is the unit of both operators, i.e., $U \cdot 1 = 1 \cdot U = U \parallel 1 = U$ for all pomsets U.

Convention 3.11. Sequential composition binds more tightly than parallel composition, i.e., $U \cdot V \parallel W$ should be read as $(U \cdot V) \parallel W$. We shall also omit parentheses due to associativity.

Many pomsets in the sequel will be either obtained by parallel or sequential composition. It therefore makes sense to identify these types of composed pomsets, as follows.

Definition 3.12 (Pomset types)**.** Let U be a pomset. We say that U is *sequential* (resp. *parallel*) if there exist non-empty pomsets U_1 and U_2 such that $U = U_1 \cdot U_2$ (resp. $U = U_1 \parallel U_2$).

Example 3.13. The pomset U from Example 3.9 is sequential, because it is the sequential composition of $\mathtt{beep}$ and $(\mathtt{tea} \parallel \mathtt{milk}) \cdot \mathtt{beep}$. It is not parallel, because its underlying labelled poset is connected. Primitive pomsets and the empty pomset are neither parallel nor sequential.

Remark 3.14. A pomset cannot be both sequential and parallel. On the other hand, there exist non-trivial pomsets that are neither parallel nor sequential; we discuss these in Section 3.2.

When we decompose a sequential pomset into components, and one of those components is sequential, we can decompose it again, continuing until we are left with non-sequential pomsets. Because sequential composition is associative, this decomposition can take place in different ways, but what matters in the end is the sequence of non-sequential pomsets obtained. We can make similar observations for parallel composition; indeed, because of commutativity of parallel composition, the order of the components does not matter either. This leads to the following.

Definition 3.15 (Factorisation)**.** Let $U \in \mathsf{Pom}$. When $U = U_1 \cdots U_n$ with each U_i non-sequential and non-empty, the sequence $U_1, \dots, U_n$ is called a *sequential factorisation* of U.

When $U = U_1 \parallel \cdots \parallel U_n$ such that the U_i are non-parallel and non-empty, the multiset $\{\!\!| U_1, \dots, U_n |\!\!\}$ is called a *parallel factorisation* of U.

Example 3.16. The pomset U from Example 3.9 has $\mathtt{beep}, \mathtt{tea} \parallel \mathtt{milk}, \mathtt{beep}$ as sequential factorisation. Because U is non-parallel, it has $\{\!\!| U |\!\!\}$ as parallel factorisation. The sequential factorisation of 1 is the empty sequence; the parallel factorisation of 1 is the empty multiset.

Sequential (resp. parallel) factorisation yields a unique sequence (resp. multiset) of components. This was proved by Grabowski [Gra81, Proposition 2] and later by Gischer [Gis88, Lemma 3.2].

Lemma 3.17. *Sequential and parallel factorisations exist uniquely.*

Convention 3.18. Keeping with algebraic nomenclature, the above allows us to refer to non-empty and non-sequential (resp. non-parallel) pomsets as sequential (resp. parallel) *primes*.

3.2 Series-parallelism

We focus on programs that can fork into multiple threads, which perform their own computations before joining together to resume execution as a single thread. This type of concurrency is known as *fork/join concurrency*. It can be described by series-parallel pomsets [Law75], as follows.

$$\begin{tikzcd}
\texttt{cup} \arrow[d] \arrow[dr] & \texttt{heat} \arrow[d] \\
\texttt{milk} & \texttt{tea}
\end{tikzcd}$$

Figure 3.2: A non-series-parallel pomset.

Definition 3.19 (Series-parallel pomsets)**.** The set of *series-parallel pomsets*, or *sp-pomsets*, denoted SP, is the smallest set satisfying the rules

$$\frac{}{1 \in \mathsf{SP}} \qquad \frac{\mathtt{a} \in \Sigma}{\mathtt{a} \in \mathsf{SP}} \qquad \frac{U, V \in \mathsf{SP}}{U \cdot V \in \mathsf{SP}} \qquad \frac{U, V \in \mathsf{SP}}{U \parallel V \in \mathsf{SP}}$$

Remark 3.20. It should be clear that the pomsets in SP built without parallel composition have a total order on their nodes, and hence correspond exactly to Σ^*, the set of words over Σ.

All pomsets we have encountered thus far are series-parallel. However, not all pomsets can be built using sequential and parallel composition. For instance, suppose we refine our description of the tea dispenser to include the action `heat`, which heats the tea, and `cup`, which drops a cup into the holder. In this case, the machine could drop a cup before dispensing milk and tea, but before tea can be dispensed it also needs to be heated (c.f. Figure 3.2). Note that there is no causal dependency between `heat` and `milk`: indeed, the milk may be poured before heating and pouring tea, or vice versa. The resulting pomset cannot be divided into parallel or sequential parts [Law78]; because it is also non-trivial, it is not series-parallel. In a sense, this shows that sp-pomsets cannot capture *message-passing concurrency* [LW00]. Different mechanisms to construct more general pomsets have been proposed [MH15; HSM$^+$16; FJS$^+$20], but these are beyond our scope.

Let us formalise the shape of such a message-passing interaction as follows.

Definition 3.21 (N-shapes, N-freedom [Law78])**.** Let $U = [\mathbf{u}]$ be a pomset. An N-*shape* in U is a quadruple $u_0, u_1, u_2, u_3 \in S_{\mathbf{u}}^4$ of distinct points such that $u_0 \leq_{\mathbf{u}} u_1$, $u_2 \leq_{\mathbf{u}} u_3$ and $u_0 \leq_{\mathbf{u}} u_3$ and there exists no other relation between them. We say that U is N-*free* if it has no N-shapes.

It is not hard to show that an N-shape cannot be created by sequential and parallel composition. As a matter of fact, the absence of shapes like this also means that we can deconstruct a pomset using sequential and parallel composition.[2] More formally, the following is known.

Theorem 3.22 [Gra81; VTL82; Gis88]**.** *A pomset is series-parallel if and only if it is* N*-free.*

[2]This is not unlike how distributive lattices can be characterised by the shapes of their sublattices [Bir48].

$$\begin{tikzcd}
\texttt{cup} \arrow[d] \arrow[rd] & \texttt{heat} \arrow[d] \arrow[ld] \\
\texttt{milk} & \texttt{tea}
\end{tikzcd}$$

Figure 3.3: Pomset subsumed by Figure 3.2.

3.3 Subsumption

On the one hand, a pomset may represent an ideal execution of a program, where actions scheduled in parallel by the program are also unordered. On the other hand, it is generally allowed for an implementation to introduce more ordering between the events; this can happen, for example, when the number of parallel actions exceeds the number of available CPUs. To reason about such a (partial) sequentialisation, we need the notion of *subsumption* [Gra81; Gis88], defined as follows.

Definition 3.23 (Subsumption)**.** Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$. We say U *is subsumed by* V, denoted $U \sqsubseteq V$, if there exists a labelled poset isomorphism from $\mathbf{v}$ to $\mathbf{u}$ that is also a bijection.

Intuitively, $U \sqsubseteq V$ should be interpreted to mean that any ordering of events that refines the one mandated by U is also a valid refinement of the order described by V. In other words, if $U \sqsubseteq V$, then U is "more sequential" than V, i.e., U contains all of the ordering of V and possibly more.

Example 3.24. Suppose the pomset U in Figure 3.2 represents the ideal behaviour of our tea dispenser, while the actual implementation waits for the tea to be heated before dispensing milk, as depicted in the pomset V in Figure 3.3. In that case, $V \sqsubseteq U$, as witnessed by the morphism that maps the event labelled by `cup` (resp. `heat`, `milk`, `tea`) in U to the corresponding event in V.

Remark 3.25. Subsumption is a preorder on pomsets. For finite pomsets it is also antisymmetric, making it a partial order; for infinite pomsets, antisymmetry does *not* hold [Ési02, Example 3.2(3)].

One easily shows that subsumption is irrelevant on empty and primitive pomsets.

Lemma 3.26. *Let $U, V \in \mathsf{Pom}$ with $U \sqsubseteq V$ or $V \sqsubseteq U$. If U is empty, then $U = V$. Furthermore, if U is primitive, i.e., $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $V = \mathtt{a}$.*

We can also relate pomset composition to subsumption. For instance, if a pomset is subsumed by a sequential pomset, then this sequential composition also appears in the subsumed pomset.

Lemma 3.27 (Separation; c.f. [BÉ96, Theorem A.9])**.** *Let $U, V \in \mathsf{Pom}$ with $U \sqsubseteq V$.*

(i) If $V = V_0 \cdot V_1$, then $U = U_0 \cdot U_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$.

(ii) If $U = U_0 \parallel U_1$, then $V = V_0 \parallel V_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$.

Decomposition is still possible when a sequential pomset is subsumed by a parallel pomset, but slightly more involved. This lemma can be thought of as the pivotal part of [Gis88, Lemma 5.6].

Lemma 3.28 (Interpolation)**.** *Let U, V, W, X be pomsets such that $U \cdot V \sqsubseteq W \parallel X$. Then there exist pomsets W_0, W_1, X_0, X_1 such that all of the following hold:*

$$W_0 \cdot W_1 \sqsubseteq W \qquad X_0 \cdot X_1 \sqsubseteq X \qquad U \sqsubseteq W_0 \parallel X_0 \qquad V \sqsubseteq W_1 \parallel X_1$$

Moreover, if W and X are series-parallel, then so are W_0, W_1, X_0 and X_1.

3.4 Pomset languages

Programs typically have more than one way of being executed — for instance, our tea dispenser may have the option of not pouring milk. We can collect the pomsets that model valid behaviours of the tea dispenser, or any other program, in a *pomset language*.

Definition 3.29 (Pomset languages)**.** A *pomset language* is a set of pomsets; a pomset language made up of sp-pomsets is referred to as a *series-parallel language*, or *sp-language* for short.

Example 3.30. To describe the pouring behaviour of the tea dispenser, we could have the language $L = \{\texttt{tea}, \texttt{tea} \parallel \texttt{milk}\}$. This language offers two behaviours: one where tea is poured, and one where tea and milk come out at the same time.

Convention 3.31. When $U \in \mathsf{SP}$, we sometimes use U to denote the pomset language $\{U\}$.

We can build pomset languages by composing other pomset languages. For instance, if $L, K \subseteq \mathsf{Pom}$ represent the possible executions of the first and second parts of a program, then any pomset in L sequentially composed with one in K may represent an execution of the program as a whole.

Definition 3.32 (Pomset language composition)**.** Let $L, K \subseteq \mathsf{Pom}$. We define the following.

$$L + K = L \cup K \qquad L \cdot K = \{U \cdot V \ : \ U \in L, V \in K\} \qquad L \parallel K = \{U \parallel V \ : \ U \in L, V \in K\}$$

Sequential composition in turn gives rise to the *Kleene closure*, as follows:

$$L^* = \bigcup_{n \in \mathbb{N}} L^n \quad \text{where} \quad L^0 = \{1\} \quad \text{and} \quad L^{n+1} = L^n \cdot L$$

Example 3.33. For the part of the tea dispenser behaviour before pouring, we could have the language $K = \{\texttt{cup} \parallel \texttt{heat}\}$. Sequentially composing this language with L as in Example 3.30, we find that $K \cdot L = \{(\texttt{cup} \parallel \texttt{heat}) \cdot \texttt{tea}, (\texttt{cup} \parallel \texttt{heat}) \cdot (\texttt{tea} \parallel \texttt{milk})\}$.

When describing program behaviour in terms of pomsets, we may want to further refine this description by looking at the behaviour that is internal to each action. A *substitution* [Gra81; Gis88] provides a pomset language representing the possible (internal) behaviour of each action over some other alphabet; we can apply a substitution to a language to include such internal actions.

Definition 3.34 (Pomset language substitution)**.** Let Δ be an alphabet. A *substitution* is a function $\zeta : \Sigma \to 2^{\mathsf{Pom}(\Delta)}$. We can lift such a substitution to $\zeta : \mathsf{Pom}(\Delta) \to 2^{\mathsf{Pom}(\Delta)}$, as follows:

$$\zeta(1) = \{1\} \qquad \zeta(U \cdot V) = \zeta(U) \cdot \zeta(V) \qquad \zeta(U \parallel V) = \zeta(U) \parallel \zeta(V)$$

When L is a series-parallel pomset language, we write $\zeta(L)$ for the pomset language $\bigcup_{U \in L} \zeta(U)$.

Example 3.35. Suppose that the action `tea` internally consists of the actions `mix` and `brew` (in that order), for mixing tea and water and brewing the tea, respectively. Furthermore, suppose `milk` has the internal behaviour of offering soy milk (the action `soy`) or dairy milk (the action `dairy`). We could then refine our model of behaviour by means of the substitution ζ where $\zeta(\texttt{tea}) = \{\texttt{mix}\cdot\texttt{brew}\}$, and $\zeta(\texttt{milk}) = \{\texttt{soy}, \texttt{dairy}\}$. If $L = \{\texttt{tea}, \texttt{tea} \parallel \texttt{milk}\}$, then we find that

$$\zeta(L) = \{\texttt{mix} \cdot \texttt{brew}, \texttt{mix} \cdot \texttt{brew} \parallel \texttt{soy}, \texttt{mix} \cdot \texttt{brew} \parallel \texttt{dairy}\}$$

3.5 Series-rational expressions

We now have everything in place to define a very rudimentary programming language, namely that of *series-rational expressions* [LW00]. On the one hand, this language allows us to specify the actions that a program with fork/join concurrency may perform, and compose them using sequential and parallel composition, as well as iteration and non-deterministic choice. On the other hand, the language lacks a mechanism to express control flow (i.e., conditional branching and iteration) or partial interleaving; we will use it as a boilerplate to add these features in Part I.

Definition 3.36 (Syntax)**.** The set of *series-rational expressions*, or *sr-expressions* for short, is denoted by $\mathcal{T}$ and generated by the grammar

$$e, f ::= 0 \mid 1 \mid \mathtt{a} \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^*$$

In addition to the primitive actions from Σ, series-rational expressions have the constants 0 and 1. For our purposes, the constant 0 represents the program without any valid (finite) behaviour, which can be thought of as the program that crashes immediately, or a program that loops indefinitely. The constant 1 represents the no-operation program, i.e., one that terminates immediately and successfully, without performing any action. The operator $+$ represents non-deterministic composition, i.e., $e+f$ is a program that either executes the program e or the program f. The operators $\cdot$ and $\parallel$ correspond to sequential and parallel composition respectively, i.e., $e \cdot f$ is the program that first executes e, followed by f, while $e \parallel f$ is the program that runs e and f in parallel. Lastly, the unary operator $(-)^*$ (called *Kleene star*) corresponds to (non-deterministic) iteration: e^* represents the behaviour of running e some (possibly zero) number of times.

Convention 3.37. Sequential composition takes precedence over parallel composition, which in turn takes precedence over non-deterministic composition. The unary operator of iteration has the highest precedence. For instance, an expression like $\mathtt{a} \cdot \mathtt{b} \parallel \mathtt{c} + \mathtt{d}^*$ should be read as $((\mathtt{a} \cdot \mathtt{b}) \parallel \mathtt{c}) + (\mathtt{d}^*)$.

Example 3.38. The behaviour of one iteration of the tea dispensing machine can be described by

$$e = (\mathtt{cup} \parallel \mathtt{heat}) \cdot ((\mathtt{dairy} + \mathtt{soy}) \parallel \mathtt{tea}) + (\mathtt{cup} \parallel \mathtt{heat}) \cdot \mathtt{tea}$$

In this description, the machines offers a choice of tea with milk (on the left) and tea without milk (on the right). In both branches, the machine pops out a cup and heats the tea, but in the left branch the further choice between dairy and soy milk is offered. The overall behaviour of the machine could then be described by e^*, which repeats the single-iteration behaviour in e.

Remark 3.39. We can also add an additional "parallel star" operator $(-)^\dagger$, sometimes denoted $(-)^{(*)}$. This operator corresponds to (non-deterministic) parallel iteration, i.e., $e^\dagger$ is the program that runs zero or more parallel copies of e. Expressions in this extended syntax are known as *series-parallel rational expressions* [LW00]. While $(-)^\dagger$ is not without merit for modelling program behaviour, we exclude it to keep our discussion simple. We return to this operator in Chapter 9.

So far, we have defined only the syntax of series-rational expressions. To attach a precise meaning, we can associate a pomset language with each expression, where each pomset is meant to represent a possible behaviour of the program. This is done by the semantics function $[\![-]\!]$.

Definition 3.40 (Semantics)**.** The function $[\![-]\!] : \mathcal{T} \to 2^{\mathsf{SP}}$ is defined inductively, as follows:

$$[\![0]\!] = \emptyset \qquad [\![\mathtt{a}]\!] = \mathtt{a} \qquad [\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!] \qquad [\![e^*]\!] = [\![e]\!]^*$$

$$[\![1]\!] = \{1\} \qquad [\![e + f]\!] = [\![e]\!] + [\![f]\!] \qquad [\![e \parallel f]\!] = [\![e]\!] \parallel [\![f]\!]$$

Convention 3.41. If $L, K \subseteq \mathsf{SP}$, then expressions like $L^* + \mathtt{a} \cdot K$ are both pomset languages and sr-expressions over 2^{SP}; in particular, $\mathcal{T} \subseteq \mathcal{T}(2^{\mathsf{SP}})$ as well as $2^{\mathsf{SP}} \subseteq \mathcal{T}(2^{\mathsf{SP}})$. The intended meaning of such expressions will always be clear from context. When $e, f \in \mathcal{T}(2^{\mathsf{SP}})$, we will write $e \doteq f$ to mean that e and f denote the same pomset language. In particular, if $e \in \mathcal{T}$, then $e \doteq [\![e]\!]$.

Given $e, f \in \mathcal{T}$, does their semantics coincide? This is not obvious, because $[\![e]\!]$ and $[\![f]\!]$ may be infinite, so we cannot compare them exhaustively. Fortunately, this problem is decidable.

Theorem 3.42 [LS14; BPS17]**.** *Let $e, f \in \mathcal{T}$. It is decidable whether $[\![e]\!] = [\![f]\!]$.*[3]

Remark 3.43. EXPSPACE is an upper complexity bound for this problem [BPS17]. Since checking equivalence of rational expressions is a special case, PSPACE is a lower bound [SM73; CKS96].

We can also reason about equivalence of sr-expressions syntactically. For instance, since parallel composition of pomsets is commutative [Gis88], $e \parallel f$ should have the same semantics as $f \parallel e$. The axioms of Kleene algebra [Koz94] and commutative Kleene algebra [Con71] apply [HMS⁺09].

Definition 3.44 (Bi-Kleene algebra)**.** A *bi-Kleene algebra congruence*, or *BKA congruence* for short, is a congruence $\approx$ on $\mathcal{T}$ w.r.t. all operators, such that for all $e, f, g \in \mathcal{T}$ the following hold:

$$e + 0 \approx e \qquad e + e \approx e \qquad e + f \approx f + e \qquad e + (f + g) \approx (f + g) + h$$

$$e \cdot (f \cdot g) \approx (e \cdot f) \cdot g \qquad e \cdot (f + g) \approx e \cdot f + e \cdot h \qquad (e + f) \cdot g \approx e \cdot g + f \cdot g$$

$$e \cdot 1 \approx e \approx 1 \cdot e \qquad e \cdot 0 \approx 0 \approx 0 \cdot e \qquad e \parallel f \approx f \parallel e \qquad e \parallel 1 \approx e \qquad e \parallel 0 \approx 0$$

$$e \parallel (f \parallel g) \approx (e \parallel f) \parallel g \qquad (e + f) \parallel g \approx e \parallel g + f \parallel g \qquad 1 + e \cdot e^* \approx e^* \approx 1 + e^* \cdot e$$

$$e + f \cdot g \lessapprox g \implies f^* \cdot e \lessapprox g \qquad e + f \cdot g \lessapprox f \implies e \cdot g^* \lessapprox f \qquad (\text{where } e \lessapprox f \iff e + f \approx f)$$

We write $\equiv$ for the smallest BKA congruence, and $e \leqq f$ whenever $e + f \equiv f$.

Remark 3.45. It is fairly easy to show that if $\approx$ is a BKA congruence, then $\lessapprox$ is a preorder on $\mathcal{T}$; indeed, it is a partial order up to $\approx$, in the sense that if $e, f \in \mathcal{T}$ such that $e \lessapprox f \lessapprox e$, then $e \approx f$. All operators are monotone w.r.t. $\lessapprox$, e.g., if $e, f, g \in \mathcal{T}$ such that $e \lessapprox f$, then $e \cdot g \lessapprox f \cdot g$.

Remark 3.46. Readers familiar with concurrent Kleene algebra will notice that the signature axiom of the *exchange law* [HMS⁺09], which allows (partial) interleaving of threads is missing; hence $\lessapprox$ is not related to $\sqsubseteq$. As a matter of fact, the exchange law can be added to the semantics and axioms using the general framework of *hypotheses*; this is the main result discussed in Chapter 5.

Of course, we should check whether the above axioms are *sound*, i.e., whether expressions related by $\equiv$ indeed yield the same behaviour. This turns out to be the case; more formally, we have:

Theorem 3.47 [LS14]**.** *$\doteq$ is a BKA congruence. In particular, if $e \equiv f$, then $[\![e]\!] = [\![f]\!]$.*

[3]We will present an alternative proof in Chapter 8.

Convention 3.48. Non-deterministic, sequential and parallel composition of series-rational expressions are associative (modulo their semantics); hence, we drop parentheses when possible.

It will happen quite often that we need to write down an sr-expression that non-deterministically chooses between a finite number of sr-expressions. Since non-deterministic choice is commutative and associative w.r.t. BKA equivalence, the order and bracketing of such an expression is immaterial. We therefore denote such an expression using generalised sum notation, i.e., when $\{e_1, \dots, e_n\} \subseteq \mathcal{T}$ we may denote $e_1 + \cdots e_n$ by $\sum_{e \in S} e$. More generally, the subscript to the sum operator may be some predicate that qualifies finitely many sr-expressions to appear in the summation.

When reasoning about such summations, it is useful to note that if $S \subseteq \mathcal{T}$ is finite and $f \in \mathcal{T}$ is such that, for every $e \in S$, it holds that $e \leqq f$, then we also have that $\sum_{e \in S} e \leqq f$. Conversely, if $\sum_{e \in S} e \leqq f$, then we have for every $e \in S$ that $e \leqq f$. Both of these facts can be proved easily from the axioms of non-deterministic choice. We will use this reasoning step implicitly in the sequel.

Example 3.49. Because of Theorem 3.47, we can use the axioms that build $\equiv$ to reason about series-rational expressions. For instance, recall the expression e that we saw in Example 3.38:

$$(\mathtt{cup} \parallel \mathtt{heat}) \cdot ((\mathtt{dairy} + \mathtt{soy}) \parallel \mathtt{tea}) + (\mathtt{cup} \parallel \mathtt{heat}) \cdot \mathtt{tea}$$

By left-distributivity of sequential composition over non-deterministic choice, we can factor out the appearances of $\mathtt{cup} \parallel \mathtt{heat}$ on the left-hand side of both operands to find that it is equivalent to

$$(\mathtt{cup} \parallel \mathtt{heat}) \cdot ((\mathtt{dairy} + \mathtt{soy}) \parallel \mathtt{tea} + \mathtt{tea})$$

Next, note that 1 is a unit of $\parallel$, and that $\parallel$ is commutative; hence, $1 \parallel \mathtt{tea} \equiv 1 \parallel \mathtt{tea} \equiv \mathtt{tea}$. Because $\equiv$ is a congruence, we can perform this substitution inside the above expression to obtain

$$(\mathtt{cup} \parallel \mathtt{heat}) \cdot ((\mathtt{dairy} + \mathtt{soy}) \parallel \mathtt{tea} + 1 \parallel \mathtt{tea})$$

Using distributivity of parallel composition over non-deterministic choice, we can factor out the parallel composition with `tea` in the second half of the expression to obtain

$$(\mathtt{cup} \parallel \mathtt{heat}) \cdot ((\mathtt{dairy} + \mathtt{soy} + 1) \parallel \mathtt{tea})$$

Arguably, this last expression is a simpler representation of the behaviour described by e.

Convention 3.50. When reasoning about equivalence w.r.t. some BKA congruence $\approx$, we will use $\equiv$ to emphasize equivalences that are true as a consequence of the BKA congruence axioms alone, instead of being particular to $\approx$. For instance, to prove $e \approx f$, we may write $e \equiv g \approx h \equiv f$.

Conversely, we could ask whether the axioms of BKA congruence are *complete*, i.e., whether all series-rational expressions that have the same behaviour can be proved equivalent using $\equiv$. Such a question is substantially harder to resolve than soundness — in this case, it can be answered positively by combining earlier results about (commutative) Kleene algebra [Con71; Koz94].

Theorem 3.51 [LS14]**.** *Let $e, f \in \mathcal{T}$. We have $e \equiv f$ if and only if $[\![e]\!] = [\![f]\!]$.*

We sometimes need to restrict our discourse to sr-expressions whose semantics does (not) include the empty pomset, similar to how Arden's rule [Ard61] applies only to languages not containing the empty word. This can be done with a simple extension of how Brzozowski [Brz64] inductively characterised rational expressions whose semantics contains the empty word, as follows.

Definition 3.52. We define $\mathcal{F}$ as the smallest subset of $\mathcal{T}$ satisfying the following rules:

$$\frac{}{1 \in \mathcal{F}} \qquad \frac{e \in \mathcal{F} \qquad f \in \mathcal{T}}{e + f \in \mathcal{F} \qquad f + e \in \mathcal{F}} \qquad \frac{e, f \in \mathcal{F}}{e \cdot f \in \mathcal{F} \qquad e \parallel f \in \mathcal{F}} \qquad \frac{e \in \mathcal{T}}{e^* \in \mathcal{F}}$$

To see that $\mathcal{F}$ indeed characterise the presence of the empty pomset, we record the following.

Lemma 3.53. *Let $e \in \mathcal{T}$. Now $e \in \mathcal{F}$ if and only if $1 \in [\![e]\!]$, which holds precisely when $1 \leqq e$.*

Convention 3.54. When Ψ is some logical assertion, we use $[\Psi]$ to denote the sr-expression 1 when Ψ is true, and 0 otherwise. For instance $[2 + 2 = 5] = 0$, while for $e \in \mathcal{F}$ we have that $[e \in \mathcal{F}] \leqq e$.

3.6 Series-rational systems

It is well-known that rational expressions correspond to a machine model in terms of finite automata [Kle56], and that commutative rational expressions can be described in terms of semi-linear spaces [Red64b; Pil70]. Indeed, the former correspondence has proved to be very useful as a way of constructing rational expressions, given a description in terms of a (finite) abstract machine; see, for instance [DKP+19]. In this section, we show that Kleene's technique (demonstrated also in [MY60; Ard61; Sal66; Con71; Bac75]) can be straightforwardly lifted to sr-expressions.

To illustrate, suppose our tea dispenser has two modes. In "normal" mode, it operates as usual, described by $e \in \mathcal{T}$; the machine may also terminate. The second, "cleaning" mode, would flush the machine. From normal mode, cleaning mode is activated by the action `clean`; here, the machine would flush using the action `flush` until done, at which point it transitions back to normal mode by the action `cleaned`. We can think of this behaviour in terms of the machine in Figure 3.4.

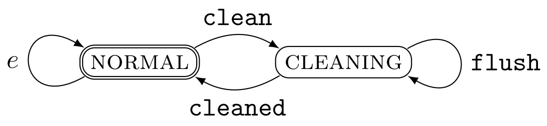


Figure 3.4: Tea dispenser modes of operation.

If we formulate the structure of this machine in terms of equations, we can use the rules for reasoning about sr-expressions to obtain an expression describing its behaviour [Ard61]. The behaviour in normal mode consists of three options: either perform the normal behaviour specified by e, followed by the behaviour in NORMAL, or emit the action `clean` and proceed with the behaviour of CLEANING, or terminate. Similarly, the cleaning mode consists of either performing the action `flush` and starting over, or emitting the action `cleaned` and returning to normal mode.

Thus, if x_{NORMAL} and x_{CLEANING} are unknowns for the expressions representing NORMAL and CLEANING respectively, then we can describe their behaviour by the following equations.

$$\begin{array}{rcrcccc} e \cdot x_{\text{NORMAL}} & + & \texttt{clean} \cdot x_{\text{CLEANING}} & + & 1 & \leqq & x_{\text{NORMAL}} \\ \texttt{cleaned} \cdot x_{\text{NORMAL}} & + & \texttt{flush} \cdot x_{\text{CLEANING}} & & & \leqq & x_{\text{CLEANING}} \end{array} \tag{3.1}$$

More precisely, the expression we are looking for is the *least* solution to x_{NORMAL} in the system above; after all, we do not want to include any behaviour that is not part of the specification.

It is not immediately obvious that the system above admits a solution to x_{NORMAL}, given its mutually recursive nature. However, we *can* find a solution, using the axioms of BKA congruence. First, we eliminate the variable x_{CLEANING}; to this end, we apply the second-to-last axiom to the second inequation in (3.1) to obtain that $\texttt{flush}^* \cdot \texttt{cleaned} \cdot x_{\text{NORMAL}} \leqq x_{\text{CLEANING}}$. We can fill in this lower bound on x_{CLEANING} into the first inequation, which tells us that

$$e \cdot x_{\text{NORMAL}} + \texttt{clean} \cdot \texttt{flush}^* \cdot \texttt{cleaned} \cdot x_{\text{NORMAL}} + 1 \leqq x_{\text{NORMAL}}$$

By right-distributivity of sequential composition over non-deterministic choice, we obtain

$$(e + \texttt{clean} \cdot \texttt{flush}^* \cdot \texttt{cleaned}) \cdot x_{\text{NORMAL}} + 1 \leqq x_{\text{NORMAL}}$$

Applying the first fixpoint axiom to the above, we can then conclude that

$$(e + \texttt{clean} \cdot \texttt{flush}^* \cdot \texttt{cleaned})^* \cdot 1 \leqq x_{\text{NORMAL}}$$

We have therefore found a lower bound on any solution to x_{NORMAL} in (3.1); plugging this back into (3.1), we also find a lower bound on x_{CLEANING}. With some additional work, it is possible to show that these lower bounds are in fact tight solutions, i.e., (3.1) holds with $\equiv$ instead of $\leqq$.

That we could solve to this system of equations is not a coincidence. In fact, it is well-known that *finite* systems of equations of rational expressions in the format of (3.1) admit a unique least solution [Ard61; Con71; Bac75]. To see this, let us first formalise the format of such systems.

Definition 3.55 (Series-rational system)**.** Let Q be a finite set. A *series-rational system* on Q, or *sr-system*, is a pair $\mathcal{S} = \langle M, b \rangle$, where $M : Q^2 \to \mathcal{T}$ and $b : Q \to \mathcal{T}$. Let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$, and $e \in \mathcal{T}$. We call $s : Q \to \mathcal{T}(\Delta)$ a $\langle \approx, e \rangle$*-solution* to $\mathcal{S}$ if for $q \in Q$:

$$b(q) \cdot e + \sum_{q' \in Q} M(q, q') \cdot s(q') \lessapprox s(q)$$

Lastly, s is the *least* $\langle \approx, e \rangle$-solution if, for every such solution s' and every $q \in Q$ we have $s(q) \lessapprox s'(q)$.

Convention 3.56. When $x : Q \to \mathcal{T}$ and $e \in \mathcal{T}$, we write x^e for the vector given by $x^e(q) = x(q) \cdot e$.

Remark 3.57. In the above, we can regard M as a Q-indexed matrix, and b and s as Q-indexed vectors [Ard61; Con71]. We can define the multiplication of a Q-indexed vector t by a Q-indexed matrix N in the usual way, that is, as the Q-indexed vector given by $(N \cdot t)(q) = \sum_{q' \in Q} N(q, q') \cdot t(q')$. If we lift $+$ and $\lessapprox$ pointwise to Q-indexed vectors, then s is a $\langle \approx, e \rangle$-solution when $b^e + M \cdot s \lessapprox s$.

Equivalently, s is a $\langle \approx, e \rangle$-solution if it is a pre-fixpoint of the function which sends a Q-vector x to $b^e + M \cdot x$ (w.r.t. $\lessapprox$). Furthermore, s is the *least* $\langle \approx, e \rangle$-solution if it is the least such pre-fixpoint (and hence the least fixpoint — see Lemma 3.59 below).

Example 3.58. Consider the series-rational system in (3.1). We can encode these inequations if we choose $Q = \{\textsc{normal}, \textsc{cleaning}\}$, and we choose the system $\mathcal{S} = \langle M, b \rangle$ on Q by setting

$$\begin{array}{lll} M(\textsc{normal}, \textsc{normal}) = e & M(\textsc{normal}, \textsc{cleaning}) = \texttt{clean} & b(\textsc{normal}) = 1 \\ M(\textsc{cleaning}, \textsc{normal}) = \texttt{cleaned} & M(\textsc{cleaning}, \textsc{cleaning}) = \texttt{flush} & b(\textsc{cleaning}) = 0 \end{array}$$

Since $\equiv$ in particular is a BKA congruence on $\mathcal{T}$, we have that any $\langle \equiv, 1 \rangle$-solution to $\mathcal{S}$ can be plugged into (3.1) on the corresponding positions for the unknowns. Conversely, any solution to the unknowns in (3.1), such as the one demonstrated above, is a $\langle \equiv, 1 \rangle$-solution to $\mathcal{S}$.

On an intuitive level, if $\mathcal{S} = \langle M, b \rangle$ is an sr-system on Q, we can think of the elements of Q as *states* of an operational description, with M as the *transitions* relation, while b describes the *halting behaviour*, i.e., the behaviour that occurs when a state decides to halt execution.

Lemma 3.59. *Let $\mathcal{S} = \langle M, b \rangle$ be an sr-system on Q, and let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$, and $e \in \mathcal{T}$. Suppose s is the least $\langle \approx, e \rangle$-solution to $\mathcal{S}$. In that case, we have for $q \in Q$:*

$$b^e(q) + \sum_{q' \in Q} M(q, q') \cdot s(q') \approx s(q)$$

On the other hand, it is not immediately obvious that for a given BKA congruence $\approx$ and $e \in \mathcal{T}$ a $\langle \approx, e \rangle$-solution even exists, let alone a least solution. Fortunately, algorithms to compute solutions and least solutions exist [Ard61; Con71; Bac75]; the solutions computed are independent of $\approx$ or e. These algorithms commonly take their cue from Kleene's method to construct a rational expression from a finite automaton [Kle56]; we present one such construction below.

Theorem 3.60. *Let $\mathcal{S} = \langle M, b \rangle$ be an sr-system on Q. We can construct an $s : Q \to \mathcal{T}$ such that, for any BKA congruence $\approx$ on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$ and any $e \in \mathcal{T}$, the Q-vector $s^e : Q \to \mathcal{T}$ is the least $\langle \approx, e \rangle$-solution to $\mathcal{S}$; we call such an s the* least solution *to $\mathcal{S}$.*

Proof. This proof is somewhat involved. We proceed by induction on the size of Q, which is possible because it is finite. In the base, where $Q = \emptyset$, we can choose for s the empty function from Q to $\mathcal{T}$.

For the inductive step, let $Q \neq \emptyset$. Our induction hypothesis is that, for any sr-system on a set strictly smaller than Q, we can construct a least solution. Let $Q = Q' \cup \{q\}$ with $q \notin Q'$. We craft the series-rational system $\mathcal{S}' = \langle M', b' \rangle$ on Q' by choosing $M' : {Q'}^2 \to \mathcal{T}$ and $b' : Q \to \mathcal{T}$ given by

$$M'(q', q'') = M(q', q'') + M(q', q) \cdot M(q, q)^* \cdot M(q, q'')$$
$$b'(q') = b(q') + M(q', q) \cdot M(q, q)^* \cdot b(q)$$

The intuition to this new system is that q is "eliminated" from the new system, by embedding its dynamics into the behaviour of the other states: transitions from q' to q'' as recorded in M' are those in M', but augmented with the possibility of first taking a detour via q, following some self-transitions there, and then transitioning to q''. Similarly, the terminating behaviour in q' as recorded in b' may also transition to q and take zero or more self-transitions, before halting there.

By induction, we obtain the least solution $s' : Q' \to \mathcal{T}$ to $\mathcal{S}'$. We choose $s : Q \to \mathcal{T}$ by setting

$$s(q') = \begin{cases} M(q,q)^* \cdot \left(b(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'(q'') \right) & q' = q \\ s'(q') & q' \neq q \end{cases}$$

Let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$, and let $e \in \mathcal{T}$. For $q' \in Q$ we then have that

$$\begin{aligned} & b^e(q') + \sum_{q'' \in Q} M(q', q'') \cdot s^e(q'') \\ \approx\ & b^e(q') + \sum_{q'' \in Q'} M(q', q'') \cdot s^e(q'') + M(q', q) \cdot s^e(q) && \text{(split out sum)} \\ \approx\ & b^e(q') + \sum_{q'' \in Q'} M(q', q'') \cdot s'^e(q'') + M(q', q) \cdot M(q, q)^* \cdot \left(b^e(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'^e(q'') \right) && (*) \end{aligned}$$

On the one hand, if $q' \in Q'$, then $\approx$ allows us to rearrange $(*)$ into the following

$$\begin{aligned}
&b^e(q') + M(q', q) \cdot M(q, q)^* \cdot b^e(q) + \sum_{q'' \in Q'} (M(q', q'') + M(q', q) \cdot M(q, q)^* \cdot M(q, q'')) \cdot s'^e(q'') \\
&\approx b'^e(q') + \sum_{q'' \in Q'} M'(q', q'') \cdot s'^e(q'') && \text{(def. } b' \text{ and } M'\text{)} \\
&\lessapprox s'^e(q') = s^e(q) && \text{(induction and def. } s\text{)}
\end{aligned}$$

On the other hand, if $q' = q$, then $(*)$ is precisely the expression below, which allows us to derive

$$\begin{aligned}
&b^e(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'^e(q'') + M(q, q) \cdot M(q, q)^* \cdot \Big(b^e(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'^e(q'')\Big) \\
&\approx (1 + M(q, q) \cdot M(q, q)^*) \cdot \Big(b^e(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'^e(q'')\Big) && \text{(distributivity)} \\
&\approx M(q, q)^* \cdot \Big(b^e(q) + \sum_{q'' \in Q'} M(q, q'') \cdot s'^e(q'')\Big) = s^e(q)
\end{aligned}$$

Thus the condition on every $q' \in Q$ is satisfied. This makes s a $\langle \approx, e\rangle$-solution to $\mathcal{S}$.

To see that it is the *least* $\langle \approx, e\rangle$-solution, suppose that $t : Q \to \mathcal{T}(\Delta)$ is a $\langle \approx, e\rangle$-solution. First, we note that we can derive the following.

$$b^e(q) + \sum_{q' \in Q'} M(q, q') \cdot t(q') + M(q, q) \cdot t(q) \approx b^e(q) + \sum_{q' \in Q} M(q, q') \cdot t(q') \lessapprox t^e(q)$$

and hence, by the fixpoint axiom, we find that

$$M(q, q)^* \cdot \Big(b^e(q) + \sum_{q' \in Q'} M(q, q') \cdot t(q')\Big) \lessapprox t(q) \tag{3.2}$$

Let $t' : Q' \to \mathcal{T}(\Delta)$ be the restriction of t to Q'. We claim that t' is a solution to $\mathcal{S}'$. To see this, we derive using the definitions of b', M' and t' for $q' \in Q'$ that

$$\begin{aligned}
&b'^e(q') + \sum_{q'' \in Q'} M'(q', q'') \cdot t'(q'') \\
&\approx b^e(q') + M(q', q) \cdot M(q, q)^* \cdot b^e(q) + \sum_{q'' \in Q'} (M(q', q'') + M(q', q) \cdot M(q, q)^* \cdot M(q, q'')) \cdot t(q'')
\end{aligned}$$

Using distributivity, $\approx$ allows us to rearrange the above to

$$b^e(q') + M(q', q) \cdot M(q, q)^* \cdot \Big(b^e(q) + \sum_{q'' \in Q'} \cdot M(q, q'') \cdot t(q'')\Big) + \sum_{q'' \in Q'} M(q', q'') \cdot t(q'')$$

Next, using eq. (3.2), we can show that the above is related by $\lessapprox$ to

$$\begin{aligned}
b^e(q') + M(q', q) \cdot t(q) + \sum_{q'' \in Q'} M(q', q'') \cdot t(q'') &\approx b^e(q') + \sum_{q'' \in Q'} M(q', q'') \cdot t(q'') \\
&\lessapprox t(q') = t'(q')
\end{aligned}$$

Since s'^e is the *least* $\langle \approx, e\rangle$-solution to $\mathcal{S}$, we have for $q' \in Q'$ that $s^e(q') = s'^e(q') \lessapprox t'(q') = t(q')$.

It remains to prove that $s^e(q) \lessapprox t(q)$. To this end, we derive using eq. (3.2) that

$$\begin{aligned} s^e(q) &= M(q,q)^* \cdot \Big(b^e(q) + \sum_{q'' \in Q'} M(q,q'') \cdot s'^e(q'')\Big) \\ &\lessapprox M(q,q)^* \cdot \Big(b^e(q) + \sum_{q'' \in Q'} M(q,q'') \cdot t'(q'')\Big) \lessapprox t(q) \end{aligned} \qquad \square$$

It is important to emphasize the wording of Theorem 3.60: if s is the constructed vector, and we choose *any* BKA congruence $\approx$ and $e \in \mathcal{T}$, then s is the least $\langle \approx, e\rangle$-solution. This is true even if $\approx$ is a BKA congruence on sr-expressions over a larger alphabet, despite s assigning states to sr-expressions over Σ. To demonstrate the power of least solutions, we record the following.

Corollary 3.61. *Let $\mathcal{S} = \langle M, b\rangle$ be an sr-system on Q with least solution s. Then $[\![-]\!] \circ s$ is the least function $L : Q \to 2^{\mathsf{SP}}$ (w.r.t. the pointwise inclusion order) such that for $q \in Q$ we have*

$$[\![b(q)]\!] \cup \bigcup_{q \in Q} [\![M(q,q')]\!] \cdot L(q') \subseteq L(q) \tag{3.3}$$

Proof. Recall the BKA congruence $\doteq$ on $\mathcal{T}(2^{\mathsf{SP}})$, as in Convention 3.41.

By Theorem 3.60, we have that s is the least $\langle \doteq, 1\rangle$-solution to $\mathcal{S}$, which means that it is the least function from Q to $\mathcal{T}(2^{\mathsf{SP}})$ such that for $q \in Q$:

$$b(q) \cdot 1 + \sum_{q \in Q} M(q,q') \cdot s(q') \mathrel{\dot{\leq}} s(q)$$

where $\dot{\leq}$ is the precongruence associated with $\doteq$. Let $L : Q \to 2^{\mathsf{SP}}$ be the least function satisfying eq. (3.3). Since L is also a function from Q to $\mathcal{T}(2^{\mathsf{SP}})$, we have $s(q) \mathrel{\dot{\leq}} L(q)$, and hence $[\![s(q)]\!] \subseteq L(q)$. Conversely, since $[\![-]\!] \circ s$ satisfies the condition on L, we find for $q \in Q$ that $L(q) \subseteq [\![s(q)]\!]$. $\square$

3.A Proof of unique factorisation lemma

To prove Lemma 3.17, we need the following generalisation of a lemma due to Levi [Lev44].

Lemma 3.A.1. *Let U, V, W, X be pomsets such that $U \cdot V = W \cdot X$. There exists a pomset Y such that either $U = W \cdot Y$ and $Y \cdot V = X$, or $U \cdot Y = W$ and $V = Y \cdot X$.*

Proof. Let $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{x}$ be labelled posets such that $U = [\mathbf{u}]$, $V = [\mathbf{v}]$, $W = [\mathbf{w}]$ and $X = [\mathbf{x}]$. Without loss of generality, we assume that $S_\mathbf{u} \cup S_\mathbf{v} = S_\mathbf{w} \cup S_\mathbf{x}$, with $S_\mathbf{u}$ and $S_\mathbf{v}$ as well as $S_\mathbf{w}$ and $S_\mathbf{x}$ disjoint.

Suppose, towards a contradiction, that $S_\mathbf{u}$ is incomparable to $S_\mathbf{w}$, i.e., that $S_\mathbf{u} \not\subseteq S_\mathbf{w}$ and $S_\mathbf{w} \not\subseteq S_\mathbf{u}$. Then there exists a $u \in S_\mathbf{u} \setminus S_\mathbf{w}$ and a $w \in S_\mathbf{w} \setminus S_\mathbf{u}$. Since $u \notin S_\mathbf{w}$, it follows that $u \in S_\mathbf{x}$;

by the same reasoning, we find that $w \in S_{\mathbf{v}}$. But then $u \leq_{\mathbf{u} \cdot \mathbf{v}} w$, and $w \leq_{\mathbf{w} \cdot \mathbf{x}} u$, and since $\leq_{\mathbf{u} \cdot \mathbf{v}}$ and $\leq_{\mathbf{w} \cdot \mathbf{x}}$ coincide, we find that $u = w$ by antisymmetry; this is a contradiction, since $u \in S_{\mathbf{u}}$ and $w \not\in S_{\mathbf{u}}$. Thus, our assumption must have been false, and hence $S_{\mathbf{u}} \subseteq S_{\mathbf{w}}$ or $S_{\mathbf{w}} \subseteq S_{\mathbf{u}}$.

For the remainder of this proof, suppose that $S_{\mathbf{u}} \subseteq S_{\mathbf{w}}$; we can prove the claim when $S_{\mathbf{w}} \subseteq S_{\mathbf{u}}$ using similar arguments. We choose the labelled poset $\mathbf{y}$ such that $S_{\mathbf{y}} = S_{\mathbf{w}} \setminus S_{\mathbf{u}}$, $\leq_{\mathbf{y}} = \leq_{\mathbf{w}} \cap S_{\mathbf{y}}^2$ and $\lambda_{\mathbf{y}} : S_{\mathbf{y}} \to \Sigma$ such that $\lambda_{\mathbf{y}}(y) = \lambda_{\mathbf{w}}(y)$. We now claim that $\mathbf{w} = \mathbf{u} \cdot \mathbf{y}$. To see this, we show that their carriers, orders and labellings coincide.

- For the carrier, note that $\mathbf{u}$ and $\mathbf{y}$ are disjoint, and that $S_{\mathbf{w}} = S_{\mathbf{u}} \cup (S_{\mathbf{w}} \setminus S_{\mathbf{u}}) = S_{\mathbf{u}} \cup S_{\mathbf{y}}$.
- For the order, suppose first that $w_0, w_1 \in S_{\mathbf{w}}$ with $w_0 \leq_{\mathbf{w}} w_1$. There are four cases.
  - If $w_0, w_1 \in S_{\mathbf{u}}$, then since $w_0 \leq_{\mathbf{w}} w_1$, we have $w_0 \leq_{\mathbf{w} \cdot \mathbf{x}} w_1$. Because $\mathbf{w} \cdot \mathbf{x} = \mathbf{u} \cdot \mathbf{v}$ it follows that $w_0 \leq_{\mathbf{u} \cdot \mathbf{v}} w_1$, and hence $w_0 \leq_{\mathbf{u}} w_1$. This in turn means that $w_0 \leq_{\mathbf{u} \cdot \mathbf{y}} w_1$.
  - If $w_0, w_1 \in S_{\mathbf{y}}$, then $w_0 \leq_{\mathbf{y}} w_1$, and thus $w_0 \leq_{\mathbf{u} \cdot \mathbf{y}} w_1$.
  - If $w_0 \in S_{\mathbf{u}}$ and $w_1 \in S_{\mathbf{y}}$, then $w_0 \leq_{\mathbf{u} \cdot \mathbf{y}} w_1$ by definition.
  - The case where $w_1 \in S_{\mathbf{u}}$ and $w_0 \in S_{\mathbf{y}}$ can be discounted, for here we find that $w_0 \in S_{\mathbf{y}} \subseteq S_{\mathbf{v}}$, and thus $w_1 \leq_{\mathbf{u} \cdot \mathbf{v}} w_0$, meaning that $w_1 \leq_{\mathbf{w} \cdot \mathbf{x}} w_0$, which in turn implies that $w_0 = w_1$, contradicting that $S_{\mathbf{u}}$ and $S_{\mathbf{y}}$ are disjoint.

  Now suppose that $w_0, w_1 \in S_{\mathbf{w}}$ with $w_0 \leq_{\mathbf{u} \cdot \mathbf{y}} w_1$. There are three cases to consider.
  - If $w_0, w_1 \in S_{\mathbf{u}}$, then $w_0 \leq_{\mathbf{u}} w_1$, and thus $w_0 \leq_{\mathbf{u} \cdot \mathbf{v}} w_1$. Since $\mathbf{u} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{x}$, we have that $w_0 \leq_{\mathbf{w} \cdot \mathbf{x}} w_1$. Since $w_0, w_1 \in S_{\mathbf{w}}$, we have $w_0 \leq_{\mathbf{w}} w_1$.
  - If $w_0, w_1 \in S_{\mathbf{y}}$, then $w_0 \leq_{\mathbf{y}} w_1$. Since $\leq_{\mathbf{y}} \subseteq \leq_{\mathbf{w}}$, we find that $w_0 \leq_{\mathbf{w}} w_1$.
  - If $w_0 \in S_{\mathbf{u}}$ and $w_1 \in S_{\mathbf{y}}$, then $w_1 \in S_{\mathbf{v}}$ and therefore $w_0 \leq_{\mathbf{u} \cdot \mathbf{v}} w_1$. Since $\mathbf{u} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{x}$, we have that $w_0 \leq_{\mathbf{w} \cdot \mathbf{x}} w_1$. Since $w_0, w_1 \in S_{\mathbf{w}}$, we then know that $w_0 \leq_{\mathbf{w}} w_1$.
- For the labelling, let $w \in S_{\mathbf{w}}$. If $w \in S_{\mathbf{u}}$, then $\lambda_{\mathbf{w}}(w) = \lambda_{\mathbf{w} \cdot \mathbf{x}}(w) = \lambda_{\mathbf{u} \cdot \mathbf{v}}(w) = \lambda_{\mathbf{u}}(w) = \lambda_{\mathbf{u} \cdot \mathbf{y}}(w)$. On the other hand, if $w \in S_{\mathbf{y}}$, then $\lambda_{\mathbf{w}}(w) = \lambda_{\mathbf{y}}(w)$ by definition of $\mathbf{y}$.

We now claim that $\mathbf{v} = \mathbf{y} \cdot \mathbf{x}$. To this end, we first note that if $s \in S_{\mathbf{x}}$, then $s \not\in S_{\mathbf{w}}$, and hence $s \not\in S_{\mathbf{u}}$ by the fact that $S_{\mathbf{u}} \subseteq S_{\mathbf{w}}$; since $s \in S_{\mathbf{u}} \cup S_{\mathbf{v}}$, it follows that $s \in S_{\mathbf{v}}$. This tells us that $S_{\mathbf{x}} \subseteq S_{\mathbf{v}}$. Furthermore, note that $s \in S_{\mathbf{y}}$ if and only if $s \in S_{\mathbf{w}}$ but $s \not\in S_{\mathbf{u}}$, which holds precisely when $s \not\in S_{\mathbf{x}}$ but $s \in S_{\mathbf{v}}$; in other words, $S_{\mathbf{y}} = S_{\mathbf{v}} \setminus S_{\mathbf{x}}$. Next, we observe that $\leq_{\mathbf{y}} = \leq_{\mathbf{w}} \cap S_{\mathbf{y}}^2 = \leq_{\mathbf{w} \cdot \mathbf{x}} \cap S_{\mathbf{y}}^2 = \leq_{\mathbf{u} \cdot \mathbf{v}} \cap S_{\mathbf{y}}^2 = \leq_{\mathbf{v}} \cap S_{\mathbf{y}}^2$, where in the last step we use that $S_{\mathbf{y}} \subseteq S_{\mathbf{v}}$.

These premises exactly mirror the premises that we used to derive that $\mathbf{u} \cdot \mathbf{y} = \mathbf{w}$ using the arguments above; a similar argument thus allows us to derive that $\mathbf{v} = \mathbf{y} \cdot \mathbf{x}$. □

We shall also need the following variant of the previous lemma for parallel composition.

Lemma 3.A.2. *Let $U, V, W, X \in \mathsf{Pom}$ with $U \parallel V = W \parallel X$. There exist $Y_0, Y_1, Z_0, Z_1 \in \mathsf{Pom}$ s.t.*

$$U = Y_0 \parallel Y_1 \qquad V = Z_0 \parallel Z_1 \qquad W = Y_0 \parallel Z_0 \qquad X = Y_1 \parallel Z_1$$

Proof. Let $U = [\mathbf{u}]$, $V = [\mathbf{v}]$, $W = [\mathbf{w}]$, and $X = [\mathbf{x}]$, and assume without loss of generality that $\mathbf{u}$ and $\mathbf{v}$ as well as $\mathbf{w}$ and $\mathbf{x}$ are disjoint, and that $\mathbf{u} \parallel \mathbf{v} = \mathbf{w} \parallel \mathbf{x}$. When $\mathbf{s}$ and $\mathbf{t}$ are labelled posets, we write $\mathbf{s} \cap \mathbf{t}$ for the labelled poset given by $S_{\mathbf{s}\cap\mathbf{t}} = S_\mathbf{s} \cap S_\mathbf{t}$, $\leq_{\mathbf{s}\cap\mathbf{t}} = \leq_\mathbf{s} \cap \leq_\mathbf{t}$ and $\lambda_{\mathbf{s}\cap\mathbf{t}}(s) = \lambda_\mathbf{s}(s)$.

We can then choose labelled posets $\mathbf{y}_0$, $\mathbf{y}_1$, $\mathbf{z}_0$ and $\mathbf{z}_1$ such that

$$\mathbf{y}_0 = \mathbf{u} \cap \mathbf{w} \qquad \mathbf{z}_0 = \mathbf{v} \cap \mathbf{w} \qquad \mathbf{y}_1 = \mathbf{u} \cap \mathbf{x} \qquad \mathbf{z}_1 = \mathbf{v} \cap \mathbf{x}$$

We claim that $\mathbf{u} = \mathbf{y}_0 \parallel \mathbf{y}_1$, $\mathbf{v} = \mathbf{z}_0 \parallel \mathbf{z}_1$, $\mathbf{w} = \mathbf{y}_0 \parallel \mathbf{z}_0$ and $\mathbf{x} = \mathbf{y}_1 \parallel \mathbf{z}_1$; the claim is then satisfied by $Y_i = [\mathbf{y}_i]$ and $Z_i = [\mathbf{z}_i]$ for $i \in \{0, 1\}$. Because the proofs are similar, we show only the first equality.

- For the carrier, we can derive

$$S_\mathbf{u} = S_\mathbf{u} \cap (S_\mathbf{w} \cup S_\mathbf{x}) = (S_\mathbf{u} \cap S_\mathbf{w}) \cup (S_\mathbf{u} \cap S_\mathbf{x}) = S_{\mathbf{y}_0} \cup S_{\mathbf{y}_1}$$

- For the ordering, first suppose that $u_0 \leq_\mathbf{u} u_1$. There are four cases to consider.
  - If $u_0, u_1 \in S_\mathbf{w}$, then $u_0 \leq_\mathbf{w} u_1$ and hence $u_0 \leq_{\mathbf{y}_0} u_1$, meaning that $u_0 \leq_{\mathbf{y}_0\parallel\mathbf{y}_1} u_1$.
  - If $u_0, u_1 \in S_\mathbf{x}$, then $u_0 \leq_\mathbf{x} u_1$ and hence $u_0 \leq_{\mathbf{y}_1} u_1$, meaning that $u_0 \leq_{\mathbf{y}_0\parallel\mathbf{y}_1} u_1$.
  - The case where $u_0 \in S_\mathbf{w}$ and $u_1 \in S_\mathbf{x}$ can be disregarded, for then we have that $u_0 \leq_{\mathbf{u}\parallel\mathbf{v}} u_1$, which means that $u_0 \leq_{\mathbf{w}\parallel\mathbf{x}} u_1$ and hence $u_0, u_1 \in S_\mathbf{w}$ or $u_0, u_1 \in S_\mathbf{x}$, contradicting that $S_\mathbf{w}$ and $S_\mathbf{x}$ are disjoint.
  - The case where $u_0 \in S_\mathbf{x}$ and $u_1 \in S_\mathbf{w}$ can be similarly disregarded.

- For the labelling, let $u \in S_\mathbf{u}$. On the one hand, if $u \in S_\mathbf{w}$, then $\lambda_\mathbf{u}(u) = \lambda_{\mathbf{u}\parallel\mathbf{v}}(u) = \lambda_{\mathbf{w}\parallel\mathbf{x}}(u) = \lambda_\mathbf{w}(u) = \lambda_{\mathbf{y}_0}(u) = \lambda_{\mathbf{y}_0\parallel\mathbf{y}_1}(u)$. On the other hand, if $u \in S_\mathbf{x}$, then $\lambda_\mathbf{u}(u) = \lambda_{\mathbf{u}\parallel\mathbf{v}}(u) = \lambda_{\mathbf{w}\parallel\mathbf{x}}(u) = \lambda_\mathbf{x}(u) = \lambda_{\mathbf{y}_1}(u) = \lambda_{\mathbf{y}_0\parallel\mathbf{y}_1}(u)$. □

This then leads to the following property of non-parallel and non-empty pomsets.

Lemma 3.A.3. *If $U_1, \dots, U_n, W, X \in \mathsf{Pom}$ with $n \geq 1$ such that $U_1, \dots, U_n$ and X are non-empty and non-parallel, and $U_1 \parallel \cdots U_n = W \parallel X$, then there exists a bijection $f : \{1, \dots, n\} \to \{1, \dots, n\}$ such that $U_{f(1)} \parallel \cdots \parallel U_{f(n-1)} = W$ and $U_{f(n)} = X$.*

Proof. We proceed by induction on n. In the base, let $n = 1$. We then have that $U_1 = W \parallel X$; since U_1 is non-empty and non-sequential, it follows that W must be empty. The claim then holds vacuously, if we choose the unique bijection from $\{1, \dots, n\} = \{1\}$ to itself.

For the inductive step, let $n > 1$. By Lemma 3.A.2, we find $Y_0, Y_1, Z_0, Z_1 \in \mathsf{Pom}$ such that

$$U_1 \parallel \cdots \parallel U_{n-1} = Y_0 \parallel Y_1 \qquad U_n = Z_0 \parallel Z_1 \qquad W = Y_0 \parallel Z_0 \qquad X = Y_1 \parallel Z_1$$

Because U_n is non-empty and non-parallel, there are two cases to distinguish.

- If $Z_0 = 1$, then Y_1 must be empty too, because otherwise $X = Y_1 \parallel U_n$ is a parallel pomset. In that case, $U_n = Z_1 = X$, and $W = Y_0$, meaning that $U_1 \parallel \cdots \parallel U_{n-1} = W$. We can choose the identity on $\{1, \dots, n\}$ to satisfy the claim.
- If $Z_1 = 1$, then $U_n = Z_0$ and $X = Y_1$, meaning $U_1 \parallel \cdots \parallel U_{n-1} = Y_0 \parallel X$ and $W = Y_0 \parallel U_n$. By induction, we find a bijection f' on $\{1, \dots, n-1\}$ such that $U_{f'(1)} \parallel \cdots \parallel U_{f'(n-2)} = Y_0$ and $U_{f'(n-1)} = X$. We choose a bijection f on $\{1, \dots, n\}$ by setting $f(i) = f'(i)$ for $1 \leq i \leq n-2$, $f(n-1) = n$ and $f(n) = f'(n-1)$. Thus, $U_{f(1)} \parallel \cdots \parallel U_{f(n-1)} = U_{f(1)} \parallel \cdots \parallel U_{f(n-2)} \parallel U_{f(n-1)} = Y_0 \parallel U_n = W$, as well as $U_{f(n)} = U_{f'(n-1)} = X$. □

We can now prove Lemma 3.17, as follows.

Lemma 3.17. *Sequential and parallel factorisations exist uniquely.*

Proof. Let $U \in \mathsf{Pom}$. To find a sequential factorisation of U, we proceed by induction on the size of U. In the base, $U = 1$; we choose $n = 0$. For the inductive step, there are two cases. First, if U is non-sequential, then choose $U_1 = U$. Otherwise, if U is sequential, then $U = U' \cdot U''$ for $U', U'' \in \mathsf{Pom}$. Since U' and U'' are strictly smaller than U, they admit sequential factorisations $U'_1, \dots, U'_{n'}$ and $U''_1, \dots, U''_{n''}$. We choose $n = n' + n''$ and for $1 \leq i \leq n'$ that $U_i = U'_i$, while for $n' < i \leq n$ that $U_i = U''_{i-n'}$. These U_i are non-sequential and non-empty, and $U = U' \cdot U'' = U_1 \cdots U_{n'} \cdot U_{n'+1} \cdots U_n$.

To see that sequential factorisations are unique, suppose that U admits sequential factorisations $U_1, \dots, U_n$ and $U'_1, \dots, U'_{n'}$. We proceed by induction on $n + n'$ to show that $n = n'$, and that for $1 \leq i \leq n$ we have $U_i = U'_i$. In the base, where $n + n' = 0$, the claim holds immediately.

For the inductive step, where $n + n' > 0$, we first note that necessarily $n, n' > 0$; after all, if $n = 0$ or $n' = 0$, then one of $U_1 \cdots U_n$ or $U'_1 \cdots U'_{n'}$ is empty, while the other one is not — which would contradict that $U_1 \cdots U_n = U = U'_1 \cdots U'_{n'}$. We thus proceed knowing that $n, n' > 0$.

Now, suppose the claim holds for smaller values of $n+n'$. Since $U_1 \cdots U_n = U'_1 \cdots U'_{n'}$, we find by Lemma 3.A.1 that there exists a $V \in \mathsf{Pom}$ such that either $U_1 = U'_1 \cdots U'_{n'-1} \cdot V$ and $V \cdot U_n = U_{n'}$, or

$U_1 \cdot V = U'_1 \cdots U'_{n'-1}$ and $U_n = V \cdot U'_{n'}$. However, since U_n and U'_n are non-sequential and non-empty, it follows that V must be empty. We then know that $U_n = U_{n'}$ and $U_1 \cdots U_{n-1} = U'_1 \cdots U'_{n'-1}$. By induction, we find that $n - 1 = n' - 1$, and for $1 \leq i \leq n-1$ that $U_i = U'_i$; the claim follows.

To show unique existence of parallel factorisations, note that we can find a parallel factorisation for any pomset inductively, just like we found a sequential factorisation. It remains to prove that parallel factorisations are unique. To this end, suppose that a pomset U has parallel factorisations $\{\!|U_1, \dots, U_n|\!\}$ and $\{\!|U'_1, \dots, U'_{n'}|\!\}$. We claim that $n = n'$, and that there exists a bijection $f : \{1, \dots, n\} \to \{1, \dots, n\}$ such that for $1 \leq i \leq n$ we have $U_i = U'_{f(i)}$.

The proof proceeds by induction on $n + n'$. In the base, where $n + n' = 0$, the claim holds vacuously. For the inductive step, where $n + n' > 0$, note that $n, n' > 0$ again. Since $U_1 \parallel \cdots \parallel U_n = U'_1 \parallel \cdots \parallel U'_{n'}$, we find by Lemma 3.A.3 that there exists a bijection $g : \{1, \dots, n-1\} \to \{1, \dots, n-1\}$ such that $U_{g(1)} \parallel \cdots \parallel U_{g(n-1)} = U'_1 \parallel \cdots \parallel U'_{n'-1}$ and $U_{g(n)} = U'_{n'}$. By induction, we then find $n-1 = n'-1$, and a bijection $f' : \{1, \dots, n-1\} \to \{1, \dots, n-1\}$ such that for $1 \leq i \leq n-1$ we have $U_{f'(g(i))} = U'_i$. We choose $f : \{1, \dots, n\} \to \{1, \dots, n\}$ by setting $f(i) = f'(g(i))$ for $1 \leq i \leq n-1$, and $f(n) = g(n)$. This makes f a bijection such that for $1 \leq i \leq n$ we have $U_{f(i)} = U'_{f(i)}$. □

3.B Proofs about subsumption

To prove Lemma 3.26, we need the following lemma.

Lemma 3.B.1. *The subsumption order is a preorder on* Pom*, and antisymmetric on finite pomsets.*

Proof. For reflexivity, we note that if $U = [\mathbf{u}]$ is a pomset, then the identity function $h : S_\mathbf{u} \to S_\mathbf{u}$ given by $h(u) = u$ can easily be shown to witness that $U \sqsubseteq U$.

For transitivity, suppose that $U, V, W \in \mathsf{Pom}$ such that $U \sqsubseteq V$ and $V \sqsubseteq W$. Without loss of generality, we can assume that $U = [\mathbf{u}]$, $V = [\mathbf{v}]$ and $W = [\mathbf{w}]$, and that $h : S_\mathbf{v} \to S_\mathbf{u}$ and $g : S_\mathbf{w} \to S_\mathbf{v}$ witness that $U \sqsubseteq V$ and $V \sqsubseteq W$ respectively. It is then straightforward to show that $h \circ g : S_\mathbf{w} \to S_\mathbf{u}$ witnesses that $U \sqsubseteq W$.

For the last claim, suppose $U, V \in \mathsf{Pom}$ are finite, with $U \sqsubseteq V$ and $V \sqsubseteq U$. Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$, let $h : S_\mathbf{v} \to S_\mathbf{u}$ and $g : S_\mathbf{u} \to S_\mathbf{v}$ witness $U \sqsubseteq V$ and $V \sqsubseteq U$. A standard argument from group theory says that there must be some $n \geq 1$ such that $(h \circ g)^n$ (that is, $h \circ g$ composed with itself n times) is the identity on $S_\mathbf{u}$. We claim that $f = g \circ (h \circ g)^{n-1} : S_\mathbf{u} \to S_\mathbf{v}$ witnesses $\mathbf{v} \cong \mathbf{u}$. First, note that f is an order- and label-preserving bijection, by construction. To see that f reflects order, let $u, u' \in S_\mathbf{u}$ with $f(u) \leq_\mathbf{v} f(u')$. In that case, $g(f(u)) \leq_\mathbf{u} g(f(u'))$. But since

$g \circ f = (h \circ g)^n$, we find $u = g(f(u)) \leq_{\mathbf{u}} g(f(u')) = u'$. We conclude that $U = V$. □

Lemma 3.26. *Let $U, V \in \mathsf{Pom}$ with $U \sqsubseteq V$ or $V \sqsubseteq U$. If U is empty, then $U = V$. Furthermore, if U is primitive, i.e., $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $V = \mathtt{a}$.*

Proof. First, suppose that $U = 1$. We then have that $U = [\mathbf{1}]$ and $V = [\mathbf{v}]$, where $\mathbf{1}$ is the unique labelled poset with the empty carrier. We now claim that $\mathbf{v} = \mathbf{1}$ and thus $V = [\mathbf{1}] = 1$; to this end, we treat the case where $U \sqsubseteq 1$; the case where $1 \sqsubseteq U$ can be argued similarly. Let $h : S_{\mathbf{1}} \to S_{\mathbf{u}}$ witness that $U \sqsubseteq 1$. Then h is a bijection from $S_{\mathbf{1}} = \emptyset$ to $S_{\mathbf{u}}$; accordingly, $S_{\mathbf{u}} = \emptyset$. In that case, the order and labelling of $\mathbf{u}$ must also coincide with $\mathbf{1}$, and the claim follows.

Second, suppose that $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$. Then $U = [\mathbf{u}]$ for some pomset with singleton carrier $S_{\mathbf{u}} = \{*\}$, with $\lambda_{\mathbf{u}}(*) = \mathtt{a}$. Furthermore, $V = [\mathbf{v}]$. We then claim that $\mathbf{u} \cong \mathbf{v}$; to this end, we treat the case where $U \sqsubseteq V$; the case where $V \sqsubseteq U$ is similar. Let $h : S_{\mathbf{v}} \to S_{\mathbf{u}}$ witness that $U \sqsubseteq V$; we claim that $V \sqsubseteq U$. Since h is a bijection from $S_{\mathbf{v}}$ to $S_{\mathbf{u}}$, it follows that $S_{\mathbf{u}} = \{\dagger\}$ is a singleton. We write h^{-1} for the unique function from $S_{\mathbf{u}}$ to $S_{\mathbf{v}}$. Now, if $u \leq_{\mathbf{u}} u'$, then $u, u' \in S_{\mathbf{u}}$ and thus $u = \dagger = u'$. Consequently, $h^{-1}(u) = h^{-1}(u')$, and thus $h^{-1}(u) \leq_{\mathbf{v}} h^{-1}(u')$. Furthermore, if $u \in S_{\mathbf{u}}$, then necessarily $(h \circ h^{-1})(u) = u$. Thus, if $u \in S_{\mathbf{u}}$, then $\lambda_{\mathbf{u}}(u) = \lambda_{\mathbf{u}}(h(h^{-1}(u))) = \lambda_{\mathbf{v}}(h^{-1}(u))$, and hence $\lambda_{\mathbf{u}} = \lambda_{\mathbf{v}} \circ h^{-1}$. It follows that $h^{-1} : S_{\mathbf{u}} \to S_{\mathbf{v}}$ is a subsumption witnessing that $V \sqsubseteq U$. We can thus conclude that $U = V$ by Lemma 3.B.1. □

Lemma 3.27 (Separation; c.f. [BÉ96, Theorem A.9])**.** *Let $U, V \in \mathsf{Pom}$ with $U \sqsubseteq V$.*

(i) If $V = V_0 \cdot V_1$, then $U = U_0 \cdot U_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$.

(ii) If $U = U_0 \parallel U_1$, then $V = V_0 \parallel V_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$.

Proof. We start with the first claim. Let U, V_0 and V_1 be as in the premise, and write $U = [\mathbf{u}]$, $V_0 = [\mathbf{v}_0]$ and $V_1 = [\mathbf{v}_1]$. Without loss of generality, we can assume that $\mathbf{v}_0$ and $\mathbf{v}_1$ are disjoint, that $S_{\mathbf{v}_0} \cup S_{\mathbf{v}_1} = S_{\mathbf{u}}$, and that the identity function $S_{\mathbf{v}_0} \cup S_{\mathbf{v}_1} \to S_{\mathbf{u}}$ is the subsumption witnessing that $\mathbf{u} \sqsubseteq \mathbf{v}_0 \cdot \mathbf{v}_1$. We then choose $\mathbf{u}_i = \mathbf{u} \cap \mathbf{v}_i$ for $i \in \{0, 1\}$, and claim that $\mathbf{u}_0 \cdot \mathbf{u}_1 = \mathbf{u}$.

- For the carrier, we already know that

$$S_{\mathbf{u}_0 \cdot \mathbf{u}_1} = S_{\mathbf{u}_0} \cup S_{\mathbf{u}_1} = (S_{\mathbf{u}} \cap S_{\mathbf{v}_0}) \cup (S_{\mathbf{u}} \cap S_{\mathbf{v}_1}) = S_{\mathbf{u}} \cap (S_{\mathbf{v}_0} \cup S_{\mathbf{v}_1}) = S_{\mathbf{u}}$$

- Now suppose that $u, u' \in S_{\mathbf{u}}$ such that $u \leq_{\mathbf{u}_0 \cdot \mathbf{u}_1} u'$. There are two cases:
  - If $u, u' \in S_{\mathbf{v}_i}$ for some $i \in \{0, 1\}$, then $u \leq_{\mathbf{v}_i} u'$, and thus $u \leq_{\mathbf{v}_0 \cdot \mathbf{v}_1} u'$, meaning $u \leq_{\mathbf{u}} u'$.

  - If $u \in S_{\mathbf{v}_0}$ and $u' \in S_{\mathbf{v}_1}$, then $u \leq_{\mathbf{v}_0 \cdot \mathbf{v}_1} u'$, and thus $u \leq_{\mathbf{u}} u'$.

  In the other direction, let $u, u' \in S_{\mathbf{u}}$ with $u \leq_{\mathbf{u}} u'$. There are three cases.

  - If $u, u' \in S_{\mathbf{v}_i}$ for some $i \in \{0, 1\}$, then $u \leq_{\mathbf{v}_i} u'$, and thus $u \leq_{\mathbf{u}_i} u'$, meaning $u \leq_{\mathbf{u}_0 \cdot \mathbf{u}_1} u'$.
  - If $u \in S_{\mathbf{v}_0} = S_{\mathbf{u}_0}$ and $u' \in S_{\mathbf{v}_1} = S_{\mathbf{u}_1}$, then $u \leq_{\mathbf{u}_0 \cdot \mathbf{u}_1} u'$ immediately.
  - The case where $u \in S_{\mathbf{u}_1}$ and $u' \in S_{\mathbf{u}_0}$ can be disregarded, for there we find $u' \leq_{\mathbf{v}_0 \cdot \mathbf{v}_1} u$, and thus $u' \leq_{\mathbf{u}} u$, meaning that $u = u'$ and contradicting disjointness of $\mathbf{u}_0$ and $\mathbf{u}_1$.

- For the labelling, let $u \in S_{\mathbf{u}}$. If $u \in S_{\mathbf{v}_i}$ for $i \in \{0, 1\}$, then we derive that

$$\lambda_{\mathbf{u}}(u) = \lambda_{\mathbf{u}_i}(u) = \lambda_{\mathbf{v}_i}(u) = \lambda_{\mathbf{v}_0 \cdot \mathbf{v}_1}(u)$$

We also claim that for $i \in \{0, 1\}$, it holds that $\mathbf{u}_i \sqsubseteq \mathbf{v}_i$, as witnessed by the identity function $S_{\mathbf{v}_i} \to S_{\mathbf{u}_i}$. To see this, let $v, v' \in S_{\mathbf{v}_i}$ be such that $v \leq_{\mathbf{v}_i} v'$. We then know that $v \leq_{\mathbf{v}_0 \cdot \mathbf{v}_1} v'$, and thus $v \leq_{\mathbf{u}} v'$ by the premise. However, since $v, v' \in S_{\mathbf{v}_i} = S_{\mathbf{u}_i}$, it follows that $v \leq_{\mathbf{u}_i} v'$.

The first claim is now satisfied by choosing $V_0 = [\mathbf{v}_0]$ and $V_1 = [\mathbf{v}_1]$. The second claim can be proved analogously; here, we split up $V = [\mathbf{v}]$ according to $U_0 = [\mathbf{u}_0]$ and $U_1 = [\mathbf{u}_1]$. □

3.C Proof of interpolation lemma

Lemma 3.28 (Interpolation)**.** *Let U, V, W, X be pomsets such that $U \cdot V \sqsubseteq W \parallel X$. Then there exist pomsets W_0, W_1, X_0, X_1 such that all of the following hold:*

$$W_0 \cdot W_1 \sqsubseteq W \qquad X_0 \cdot X_1 \sqsubseteq X \qquad U \sqsubseteq W_0 \parallel X_0 \qquad V \sqsubseteq W_1 \parallel X_1$$

Moreover, if W and X are series-parallel, then so are W_0, W_1, X_0 and X_1.

Proof. Let $U = [\mathbf{u}]$, $V = [\mathbf{v}]$, $W = [\mathbf{w}]$ and $X = [\mathbf{x}]$, and assume without loss of generality that $S_{\mathbf{u}}$ and $S_{\mathbf{v}}$ are disjoint, as are $S_{\mathbf{w}}$ and $S_{\mathbf{x}}$, and that $S_{\mathbf{u}} \cup S_{\mathbf{v}} = S_{\mathbf{w}} \cup S_{\mathbf{x}}$, where $[\mathbf{u} \cdot \mathbf{v}] \sqsubseteq [\mathbf{w} \parallel \mathbf{x}]$ is witnessed by the identity $i : S_{\mathbf{w}} \cup S_{\mathbf{x}} \to S_{\mathbf{u}} \cup S_{\mathbf{v}}$. We choose labelled posets $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{x}_0$ and $\mathbf{x}_1$:

$$\mathbf{w}_0 = \mathbf{w} \cap \mathbf{u} \qquad \mathbf{w}_1 = \mathbf{w} \cap \mathbf{v} \qquad \mathbf{x}_0 = \mathbf{x} \cap \mathbf{u} \qquad \mathbf{x}_1 = \mathbf{x} \cap \mathbf{v}$$

One easily verifies that these are pairwise disjoint. To show that $[\mathbf{u}] \sqsubseteq [\mathbf{w}_0 \parallel \mathbf{x}_0]$, first note that

$$S_{\mathbf{w}_0 \parallel \mathbf{x}_0} = S_{\mathbf{w}_0} \cup S_{\mathbf{x}_0} = (S_{\mathbf{u}} \cap S_{\mathbf{w}}) \cup (S_{\mathbf{u}} \cap S_{\mathbf{x}}) = S_{\mathbf{u}} \cap (S_{\mathbf{w}} \cup S_{\mathbf{x}}) = S_{\mathbf{u}} \cap (S_{\mathbf{u}} \cup S_{\mathbf{v}}) = S_{\mathbf{u}}$$

We now claim that $i : S_{\mathbf{w}_0 \parallel \mathbf{x}_0} \to S_{\mathbf{u}}$, i.e., the identity on $S_{\mathbf{u}}$, is a subsumption witnessing that $[\mathbf{u}] \sqsubseteq [\mathbf{w}_0 \parallel \mathbf{x}_0]$. To see this, let $u_0, u_1 \in S_{\mathbf{u}}$ be such that $u_0 \leq_{\mathbf{w}_0 \parallel \mathbf{x}_0} u_1$. If $u_0 \leq_{\mathbf{w}_0} z$, then $u_0 \leq_{\mathbf{w}} u_1$

by choice of $\mathbf{w}_0$. But then $u_0 \leq_{\mathbf{w} \parallel \mathbf{x}} u_1$, and thus $u_0 \leq_{\mathbf{u} \cdot \mathbf{v}} u_1$ by the premise. Since $u_0, u_1 \in S_{\mathbf{u}}$, we can conclude that $u_0 \leq_{\mathbf{u}} u_1$. We can similarly show that $u_0 \leq_{\mathbf{u}} u_1$ when $u_0 \leq_{\mathbf{x}_0} z$ and therefore we conclude that $[\mathbf{u}] \sqsubseteq [\mathbf{w}_0 \parallel \mathbf{x}_0]$. A similar argument shows that $[\mathbf{v}] \sqsubseteq [\mathbf{w}_1 \parallel \mathbf{x}_1]$.

To see $[\mathbf{w}_0 \cdot \mathbf{w}_1] \sqsubseteq [\mathbf{w}]$, first note that $S_{\mathbf{w}_0 \cdot \mathbf{w}_1} = S_{\mathbf{w}}$ by reasoning similar to the above. We claim that $i : S_{\mathbf{w}} \to S_{\mathbf{w}_0 \cdot \mathbf{w}_1}$, i.e., the identity on $S_{\mathbf{w}}$, is a subsumption witnessing $[\mathbf{w}_0 \cdot \mathbf{w}_1] \sqsubseteq [\mathbf{w}]$. To see this, suppose that $w_0, w_1 \in S_{\mathbf{w}}$ such that $w_0 \leq_{\mathbf{w}} w_1$. Then we know that $w_0 \leq_{\mathbf{w} \parallel \mathbf{x}} w_1$, and thus $w_0 \leq_{\mathbf{u} \cdot \mathbf{v}} w_1$ by the premise. We exclude the case where $w_1 \in S_{\mathbf{u}}$ and $w_0 \in S_{\mathbf{v}}$, for then $w_1 \leq_{\mathbf{u} \cdot \mathbf{v}} w_0$ and thus $w_0 = w_1$ by antisymmetry, contradicting that $\mathbf{u}$ and $\mathbf{v}$ are disjoint. Three cases remain.

- If $w_0, w_1 \in S_{\mathbf{u}}$, then $w_0 \leq_{\mathbf{w}_0} w_1$, and thus $w_0 \leq_{\mathbf{w}_0 \cdot \mathbf{w}_1} w_1$.
- If $w_0, w_1 \in S_{\mathbf{v}}$, then $w_0 \leq_{\mathbf{w}_1} w_1$, and thus $w_0 \leq_{\mathbf{w}_0 \cdot \mathbf{w}_1} w_1$.
- If $w_0 \in S_{\mathbf{u}}$ and $w_1 \in S_{\mathbf{v}}$, then $w_0 \in S_{\mathbf{w}_0}$ and $w_1 \in S_{\mathbf{w}_1}$, thus $w_0 \leq_{\mathbf{w}_0 \cdot \mathbf{w}_1} w_1$ by definition.

Since $w_0 \leq_{\mathbf{w}_0 \cdot \mathbf{w}_1} w_1$ in all possible cases, we conclude that i preserves ordering and is therefore a subsumption. The proof that $[\mathbf{x}_0 \cdot \mathbf{x}_1] \sqsubseteq [\mathbf{x}]$ is similar.

We can now choose $W_0 = [\mathbf{w}_0]$, $W_1 = [\mathbf{w}_1]$, $X_0 = [\mathbf{x}_0]$ and $X_1 = [\mathbf{x}_1]$ to satisfy the claim. Moreover, we note that if W and X are series-parallel, then they are N-free by Theorem 3.22. The labelled posets $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{x}_0$ and $\mathbf{x}_1$ must then also be N-free, and therefore W_0, W_1, X_0 and X_1 are series-parallel by Theorem 3.22. This concludes the proof. $\square$

3.D Proofs about sr-expressions

Lemma 3.53. *Let $e \in \mathcal{T}$. Now $e \in \mathcal{F}$ if and only if $1 \in [\![e]\!]$, which holds precisely when $1 \leqq e$.*

Proof. We prove that the first implies the second, the second the third, and the third the first.

- First, suppose $e \in \mathcal{F}$. We prove that $1 \leqq e$ by induction on $\mathcal{F}$. In the base, we have two cases. On the one hand, if $e = 1$, then the claim holds immediately. On the other hand, if $e = e_0^*$, then $1 \leqq 1 + e_0 \cdot e_0^* \equiv e_0^*$, and so the claim follows.

  For the inductive step, there are two cases to consider. On the one hand, if $e = e_0 + e_1$ such that $e_i \in \mathcal{F}$ for some $i \in \{0, 1\}$, then $1 \leqq e_i$ by induction. Since $e_i \leqq e_0 + e_1 = e$, the claim then follows. On the other hand, if $e = e_0 \cdot e_1$ or $e = e_0 \parallel e_1$ such that $e_0, e_1 \in \mathcal{F}$, then $1 \leqq e_0$ and $1 \leqq e_1$ by induction. Since $1 \equiv 1 \cdot 1 \leqq e_0 \cdot e_1 = 1$, the claim then follows.

- Next, if $1 \leqq e$, then $1 \in [\![1]\!] \subseteq [\![e]\!]$ by Theorem 3.47.

- Lastly, if $1 \in \llbracket e \rrbracket$, then we prove $e \in \mathcal{F}$ by induction on e. In the base, it must be that $e = 1$; we then find that $e \in \mathcal{F}$ immediately. For the inductive step, there are five cases to consider.
  - If $e = e_0 + e_1$, then $1 \in \llbracket e_0 \rrbracket$ or $1 \in \llbracket e_1 \rrbracket$. By induction, we then find that $1 \in e_0$ or $1 \leqq e_1$, which implies that $e \in \mathcal{F}$.
  - If $e = e_0 \cdot e_1$, then there exist $U_0 \in \llbracket e_0 \rrbracket$ and $U_1 \in \llbracket e_1 \rrbracket$ with $U_0 \cdot U_1 = 1$. It then follows that $U_0 = U_1 = 1$. By induction, we then find $e_0 \in \mathcal{F}$ and $e_1 \in \mathcal{F}$, and thus $e \in \mathcal{F}$.
  - If $e = e_0 \parallel e_1$, then an argument similar to the previous case applies.
  - If $e = e_0^*$, then $e \in \mathcal{F}$ immediately. □

Lemma 3.59. *Let $\mathcal{S} = \langle M, b \rangle$ be an sr-system on Q, and let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$, and $e \in \mathcal{T}$. Suppose s is the least $\langle \approx, e \rangle$-solution to $\mathcal{S}$. In that case, we have for $q \in Q$:*

$$b^e(q) + \sum_{q' \in Q} M(q, q') \cdot s(q') \approx s(q)$$

Proof. As announced, the proof is a fairly standard argument from fixpoint theory. Let s be the least $\langle \approx, e \rangle$-solution to $\mathcal{S}$, and choose $s' : Q \to \mathcal{T}$ as follows:

$$s'(q) = b^e(q) + \sum_{q' \in Q} M(q, q') \cdot s(q')$$

Now, since s is a $\langle \approx, e \rangle$-solution to $\mathcal{S}$, we have for $q \in Q$ that $s'(q) \lessapprox s(q)$. We also claim that s' is a solution to $\mathcal{S}$. To see this, note that for $q \in Q$ we have

$$b^e(q) + \sum_{q' \in Q} M(q, q') \cdot s'(q) \lessapprox b^e(q) + \sum_{q' \in Q} M(q, q') \cdot s(q) = s'(q)$$

Since s is the *least* $\langle \approx, e \rangle$-solution to $\mathcal{S}$, it follows that for $q \in Q$ we have $s(q) \lessapprox s'(q)$. □

Part I

Algebra

Chapter 4

Hypotheses for concurrency

In the previous chapter, we discussed series-rational expressions, a simple language for specifying and verifying the order of events in programs with fork/join concurrency. Because the axioms of BKA congruence are rather general, they may also be valid for other programming languages. We can use them to reason about equivalence in those languages, as well. On the other hand, languages may have properties that are specific to their purpose and domain. Hence, the BKA congruence may not always be enough to prove the desired equivalences between actual programs.

For instance, consider the toy programming language COUNT, whose sole purpose is to increment some counter. Programs in COUNT are formed by series-rational expressions over the alphabet $\Sigma = \{\texttt{inc}(n) \; : \; n \in \mathbb{N}\}$. Here, $\texttt{inc}(n)$ is a primitive action meaning *increment the counter by* n. Conceivably, the behaviour of running $\texttt{inc}(n)$ followed by $\texttt{inc}(m)$ should contain the behaviour of running $\texttt{inc}(n+m)$. Similarly, the semantics of the no-operation program 1 should contain the semantics of incrementing by zero. If we assume these two properties of the language, we are able to prove new things about our language, such as that the semantics of $\texttt{inc}(1)^*$ includes that of $\texttt{inc}(n)$ for all natural numbers n. Nevertheless, neither of these properties are guaranteed by the relation $\leqq$ on series-rational expressions as defined in Chapter 3. We thus need a way to augment the BKA congruence with a set of additional, language-specific *hypotheses* [Coh94], as follows.

Definition 4.1 (Hypotheses)**.** A *hypothesis* is an inequation $e \leq f$ where $e, f \in \mathcal{T}$. When H is a set of hypotheses, we write $\equiv^H$ for the smallest BKA congruence on $\mathcal{T}$ that satisfies the containments in H. More concretely, whenever $e \leq f \in H$, also $e + f \equiv^H f$. Analogous to previous notation, we will abbreviate the latter by writing $e \leqq^H f$.

Remark 4.2. It is not hard to show that $\leqq^H$ is a preorder on $\mathcal{T}$; in fact, it is a partial order up to $\equiv^H$, i.e., if $e \leqq^H f \leqq^H e$, then $e \equiv^H f$. Furthermore, all operators are monotone w.r.t. $\leqq^H$.

Convention 4.3. Hypotheses are formulated as containments rather than equivalences to ease our development later on. Equivalences can still be encoded by including both $e \leq f$ and $f \leq e$ in H. Going forward, we write $\{e = f \ : \ \cdots\}$ for the hypotheses given by $\{e \leq f \ : \ \cdots\} \cup \{f \leq e \ : \ \cdots\}$.

Example 4.4. Consider the programming language COUNT outlined above, given by sr-expressions over $\{\mathtt{inc}(n) \ : \ n \in \mathbb{N}\}$. To refine our reasoning about COUNT, we choose the following hypotheses:

$$\mathsf{count} = \{\mathtt{inc}(0) \leq 1\} \cup \{\mathtt{inc}(n+m) \leq \mathtt{inc}(n) \cdot \mathtt{inc}(m) \ : \ n, m \in \mathbb{N}\}$$

To prove that, for $n \in \mathbb{N}$, it holds that $\mathtt{inc}(n) \leqq^{\mathsf{count}} \mathtt{inc}(1)^*$, we proceed by induction on n. In the base, where $n = 0$, we have that $\mathtt{inc}(0) \leq 1 \in \mathsf{count}$ implies $\mathtt{inc}(0) \leqq^{\mathsf{count}} 1$. Moreover, because $1 + \mathtt{inc}(1) \cdot \mathtt{inc}(1)^* \equiv \mathtt{inc}(1)^*$, we can conclude that $\mathtt{inc}(0) \leqq^{\mathsf{count}} \mathtt{inc}(1)^*$.

For the inductive step, suppose the claim holds for n. Then we find that

$$\mathtt{inc}(n+1) \leqq^{\mathsf{count}} \mathtt{inc}(1) \cdot \mathtt{inc}(n) \leqq^{\mathsf{count}} \mathtt{inc}(1) \cdot \mathtt{inc}(1)^* \leqq^{\mathsf{count}} 1 + \mathtt{inc}(1) \cdot \mathtt{inc}(1)^* \equiv^{\mathsf{count}} \mathtt{inc}(1)^*$$

in which the second step is a consequence of the induction hypothesis and monotonicity.

Remark 4.5. Unlike the axioms that generate $\equiv$, hypotheses are not necessarily stable under substitution, i.e., if $H = \{\mathtt{a} \leq \mathtt{b}\}$ for $\mathtt{a}, \mathtt{b} \in \Sigma$, this means that $\mathtt{a} \leqq^H \mathtt{b}$, but not necessarily $\mathtt{b} \leqq^H \mathtt{a}$.

Hypotheses have been studied extensively for rational expressions [Coh94; Koz96; Koz02; KM14; Mam15; DKP+19]. In this chapter, we take a method to synthesise a semantics of rational expressions that is sound w.r.t. a set of hypotheses [DKP+19], and extend it to sr-expressions. This extension is non-trivial because of the two-dimensional nature of sp-pomsets. Next, we study the meta-theory of hypotheses on sr-expressions, and obtain a number of tools that can help ease the recovery of decidability and completeness results under hypotheses. In particular, we propose formalisms to compose and reuse existing results concerning decidability and completeness.

4.1 Soundness

The augmented congruence on series-rational expressions is not necessarily sound w.r.t. the semantics in terms of series-rational languages introduced in the previous chapter. For instance, if count is as in Example 4.4, then $[\![1]\!]$ contains just the empty pomset, and $[\![\mathtt{inc}(0)]\!]$ is the singleton containing just the primitive pomset $\mathtt{inc}(0)$, despite the fact that $\mathtt{inc}(0) \leqq^{\mathsf{count}} 1$.

In this section, we discuss a method that can, given *any* set of hypotheses, construct a semantics of series-rational expressions in terms of pomset languages that is sound under those hypotheses. The benefit of this method is twofold. On the one hand, it allows us to synthesise semantics of a programming language intended to satisfy properties encoded as hypotheses. While the resulting pomset languages may be unwieldy at first, we shall see that in some cases an intuitive understanding can be obtained. On the other hand, the constructed semantics can be used as a proxy for falsifying properties — after all, if two programs are provably equal, then their semantics should also coincide. In Section 4.2, we shall see some techniques that, for certain hypotheses allow us to prove the converse: if two expressions coincide semantically, they are also provably equal.

The idea behind this method is to take the original semantics of series-rational expressions in terms of pomset languages, and use the hypotheses to saturate them [DKP+19]. For instance, given that $\texttt{inc}(n) \leqq^{\mathsf{count}} \texttt{inc}(1)^*$ for all $n \in \mathbb{N}$, we need to find a way to add the pomset $\texttt{inc}(n)$ for every $n \in \mathbb{N}$ to the semantics of $\texttt{inc}(1)^*$. More generally, this should work for all provable containments, i.e., since $\texttt{inc}(42) \parallel \texttt{inc}(91) \leqq^{\mathsf{count}} \texttt{inc}(1)^* \parallel \texttt{inc}(1)^*$ by monotonicity, the semantics of the latter should include the semantics of the former. To reckon with this, we introduce pomset contexts.

Definition 4.6 (Pomset contexts)**.** Let $\square \notin \Sigma$. The set of *series-parallel pomset contexts*, denoted $\mathsf{PC}^{\mathsf{sp}}$, is the smallest subset of $\mathsf{Pom}(\Sigma \cup \{\square\})$ satisfying the following rules of inference:

$$\frac{}{\square \in \mathsf{PC}^{\mathsf{sp}}} \qquad \frac{V \in \mathsf{SP} \qquad C \in \mathsf{PC}^{\mathsf{sp}}}{V \cdot C \in \mathsf{PC}^{\mathsf{sp}}} \qquad \frac{C \in \mathsf{PC}^{\mathsf{sp}} \qquad V \in \mathsf{SP}}{C \cdot V \in \mathsf{PC}^{\mathsf{sp}}} \qquad \frac{V \in \mathsf{SP} \qquad C \in \mathsf{PC}^{\mathsf{sp}}}{V \parallel C \in \mathsf{PC}^{\mathsf{sp}}}$$

Intuitively, $\square$ is a placeholder or gap where another pomset can be inserted, and an sp-pomset context is an sp-pomset with one and only one occurrence of $\square$. Given a pomset context $C \in \mathsf{PC}^{\mathsf{sp}}$ and a pomset $U \in \mathsf{Pom}$, we can "plug" U into the gap left in C to obtain the pomset $C[U] \in \mathsf{Pom}$. This new pomset contains all of the events of C except the one labelled by $\square$, which is where the events from U have been filled in; these events are ordered w.r.t. the events in C in the same way that $\square$ was. More formally, this plugging operation is done as follows.

Definition 4.7 (Context plugging)**.** Let $C \in \mathsf{PC}^{\mathsf{sp}}$ and $U \in \mathsf{Pom}$. We write $C[U]$ for the pomset defined by induction on the structure of C, as follows:

$$\square[U] = U \qquad (V \cdot C)[U] = V \cdot C[U] \qquad (C \cdot V)[U] = C[U] \cdot V \qquad (V \parallel C)[U] = V \parallel C[U]$$

If $L \subseteq \mathsf{Pom}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$, we write $C[L]$ for the pomset language $\{C[U] \ : \ U \in L\}$.

Example 4.8. We can construct the series-parallel pomset context $(\texttt{inc}(0) \parallel \texttt{inc}(1)) \cdot (\texttt{inc}(2) \parallel \square)$, and plug in $\texttt{inc}(3) \parallel \texttt{inc}(4)$ to obtain $(\texttt{inc}(0) \parallel \texttt{inc}(1)) \cdot (\texttt{inc}(2) \parallel \texttt{inc}(3) \parallel \texttt{inc}(4))$; c.f. Figure 4.1.

$$\begin{tikzcd}
\mathtt{inc}(0) \arrow[d] \arrow[dr] & \mathtt{inc}(1) \arrow[d] \arrow[dl] \\
\mathtt{inc}(2) & \square
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathtt{inc}(0) \arrow[dl] \arrow[dr] \arrow[drrr] & & \mathtt{inc}(1) \arrow[dlll] \arrow[dl] \arrow[dr] & \\
\mathtt{inc}(2) & & \mathtt{inc}(3) & & \mathtt{inc}(4)
\end{tikzcd}$$

Figure 4.1: An sp-pomset context and the pomset obtained by plugging in $\mathtt{inc}(3) \parallel \mathtt{inc}(4)$.

Using contexts, we can post-process the semantics of series-parallel pomsets using a given set of hypotheses. Intuitively, this is done by finding parts of the pomset language that (under a context) match the right side of a hypothesis, and adding the language corresponding to the left side of the hypothesis (under the same context) if this is the case. Formally, we define the following operation.

Definition 4.9 (Closure; c.f. [DKP$^+$19, Definition 2])**.** Let H be a set of hypotheses, and $L \subseteq \mathsf{Pom}$. We define the H*-closure* of L, written L^H, as the smallest language containing L, and satisfying

$$\frac{e \leq f \in H \qquad C \in \mathsf{PC}^{\mathsf{sp}} \qquad C[\llbracket f \rrbracket] \subseteq L^H}{C[\llbracket e \rrbracket] \subseteq L^H}$$

Example 4.10. Recall the set of hypotheses count from Example 4.4. Let $e = \mathtt{inc}(1)^* \parallel \mathtt{inc}(1)^*$. If we then choose $C = \square \parallel \mathtt{inc}(1)$, we find that $C[\llbracket \mathtt{inc}(1) \cdot \mathtt{inc}(1) \rrbracket] \subseteq \llbracket e \rrbracket \subseteq \llbracket e \rrbracket^{\mathsf{count}}$. Hence, because $\mathtt{inc}(2) \leq \mathtt{inc}(1) \cdot \mathtt{inc}(1) \in \mathsf{count}$, it follows that $\{\mathtt{inc}(2) \parallel \mathtt{inc}(1)\} = C[\llbracket \mathtt{inc}(2) \rrbracket] \subseteq \llbracket e \rrbracket^{\mathsf{count}}$, by the rule above. We can repeat this process to show that $\mathtt{inc}(n) \parallel \mathtt{inc}(m) \in \llbracket e \rrbracket^{\mathsf{count}}$ for all $n, m \in \mathbb{N}$, which matches that we have that $\mathtt{inc}(n) \parallel \mathtt{inc}(m) \leqq^{\mathsf{count}} \mathtt{inc}(1)^* \parallel \mathtt{inc}(1)^*$ for all $n, m \in \mathbb{N}$.

The example above already hints that H-closure can help saturate the semantics of an sr-expression such that facts derived using the hypotheses are also true in the semantics. To prove this in full generality, we first note that $(-)^H$ truly is a closure operator.

Lemma 4.11 (c.f. [DKP$^+$19, Lemma 1])**.** *Let $L, K \subseteq \mathsf{Pom}$. Then $L \subseteq K^H$ if and only if $L^H \subseteq K^H$.*

Remark 4.12. Closure with respect to a set of hypotheses is not, in general, a Kuratowski closure operator [Kur22], since it fails to commute with union. For instance, if $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $H = \{\mathtt{a} \leq \mathtt{b} + \mathtt{c}\}$, then $\{\mathtt{b}\}^H \cup \{\mathtt{c}\}^H = \{\mathtt{b}, \mathtt{c}\}$, while $\mathtt{a} \in (\{\mathtt{b}\} \cup \{\mathtt{c}\})^H$.

Lemma 4.11 then allows us to prove properties relating closure to pomset language composition.

Lemma 4.13 (c.f. [DKP$^+$19, Lemma 2])**.** *Let $L, K \subseteq \mathsf{Pom}$. The following hold:*

$$(L \cup K)^H = \left(L^H \cup K^H\right)^H \qquad (L \cdot K)^H = \left(L^H \cdot K^H\right)^H$$

$$(L \parallel K)^H = \left(L^H \parallel K^H\right)^H \qquad (L^*)^H = \left(\left(L^H\right)^*\right)^H$$

With these properties in hand, we can then show that H-closure gives rise to a semantics of series-rational language that is sound with respect to the equivalence $\equiv$ built from H.

Theorem 4.14 (c.f. [DKP+19, Theorem 2]). *If $e \equiv^H f$, then $[\![e]\!]^H = [\![f]\!]^H$.*

We conclude this section by showing how this general soundness theorem may be used to derive a base semantics of an abstract programming language, given the properties it should satisfy.

Example 4.15. Consider the programming language UNDO, whose syntax is given by rational expressions over an alphabet Σ. The purpose of UNDO is that every primitive action $\mathtt{a} \in \Sigma$ has an opposite action $\mathtt{a}^{-1} \in \Sigma$, which completely reverts $\mathtt{a}$. This means that running $\mathtt{a}$ and $\mathtt{a}^{-1}$ in sequence is the same as not doing anything. Naturally, the inverse of an inverse action is the original action, i.e., $(\mathtt{a}^{-1})^{-1} = \mathtt{a}$. We can encode this behaviour using hypotheses, as follows:

$$\mathsf{undo} = \left\{\mathtt{a} \cdot \mathtt{a}^{-1} = 1 \ : \ \mathtt{a} \in \Sigma\right\}$$

If we use $[\![-]\!]^{\mathsf{undo}}$ as a semantics, we can obtain some rather bulky pomset languages. For instance, $[\![1]\!]^{\mathsf{undo}}$ includes all pomsets where $\mathtt{a}$ is repeated n times, followed by $\mathtt{a}^{-1}$ repeated n times.

We can, however, recover a sensible semantics from $[\![-]\!]^{\mathsf{undo}}$, by pruning the redundancy. More precisely, let $M = \left\{C[\mathtt{a} \cdot \mathtt{a}^{-1}] \ : \ C \in \mathsf{PC}^{\mathsf{sp}}\right\}$ be the set of pomsets over Σ in which, for every $\mathtt{a} \in \Sigma$, there is a node labelled $\mathtt{a}^{-1}$ directly succeeding it, i.e., where some action is undone. Now, for $e \in \mathcal{T}$, the language $[\![e]\!]^{\mathsf{undo}} \setminus M$ contains all of the pomsets that record actions performed and not reverted. Indeed, it is not hard to show that this new semantics is isomorphic to the old semantics, i.e., for $e, f \in \mathcal{T}$ we have that $[\![e]\!]^{\mathsf{undo}} = [\![f]\!]^{\mathsf{undo}}$ if and only if $[\![e]\!]^{\mathsf{undo}} \setminus M = [\![f]\!]^{\mathsf{undo}} \setminus M$.

4.2 Reduction

Suppose we have a language whose semantics is sound w.r.t. the BKA congruence, as well as an additional set of hypotheses H. The question then arises: can we mechanically verify whether two programs are provably equivalent according to $\equiv^H$? Furthermore, is $\equiv^H$ sufficient to prove equivalence of expressions identified by $[\![-]\!]^H$? In this section, we try to answer these questions. By Theorem 4.14, we know that if $[\![e]\!]^H = [\![f]\!]^H$ does *not* hold, then neither can $e \equiv^H f$. Thus, to obtain a complete decision procedure, we need to answer following two questions:

(DECIDABILITY) Can we decide whether $[\![e]\!]^H = [\![f]\!]^H$?

(COMPLETENESS) Does $[\![e]\!]^H = [\![f]\!]^H$ imply $e \equiv^H f$?

In general, either of these properties may not hold, as witnessed by the following.

Example 4.16 (Due to E. Cohen, c.f. [Koz96])**.** In *Post's correspondence problem* (*PCP*) [Pos46] we are provided $x_1, \ldots, x_n, y_1, \ldots, y_n \in \{\mathtt{a}, \mathtt{b}\}^*$, and asked whether there exist $1 \leq i_1, \ldots, i_m \leq n$ with $m \geq 1$ such that $x_{i_1} \cdots x_{i_m} = y_{i_1} \cdots y_{i_m}$. It is well-known that this problem is undecidable.

Given an instance of PCP as above, we can produce $e, f \in \mathcal{T}$ and a finite set of hypotheses H such that we have a no-instance if and only if $[\![e]\!] \subseteq [\![f]\!]^H$. Since the latter can be decided by checking $[\![e+f]\!]^H = [\![f]\!]^H$, checking semantic equivalence under closure is undecidable.

We encode choice of indices by $e = (x_1 \cdot y'_1 + \cdots + x_n \cdot y'_n)^+$ where we use g^+ as shorthand for $g \cdot g^*$, and y'_i denotes the word y_i where every $\mathtt{a}$ (resp. $\mathtt{b}$) is replaced by $\mathtt{a}'$ (resp. $\mathtt{b}'$). A pomset in $[\![e]\!]$ consists of unprimed letters forming $x_{i_1} \cdots x_{i_m}$, and primed letters which form $y_{i_1} \cdots y_{i_m}$.

Next, think about what it means for $x_{i_1} \cdots x_{i_m}$ to be different from $y_{i_1} \cdots y_{i_m}$: after some common prefix w, either the former ends but the latter does not (or vice versa) or the former contains an $\mathtt{a}$ while the latter contains a $\mathtt{b}$ (or vice versa). We encode these possibilities as follows:

$$f = (\mathtt{a} \cdot \mathtt{a}' + \mathtt{b} \cdot \mathtt{b}')^* \cdot \Big((\mathtt{a} + \mathtt{b})^+ + (\mathtt{a}' + \mathtt{b}')^* + (\mathtt{a} \cdot \mathtt{b}' + \mathtt{b} \cdot \mathtt{a}') \cdot (\mathtt{a} + \mathtt{b} + \mathtt{a}' + \mathtt{b}')^* \Big)$$

Again, every element of $[\![f]\!]$ contains two pomsets: one formed by the unprimed letters, and one formed by the primed letters (with their primes removed); the construction of f guarantees that these words differ. In this encoding, the relative ordering of primed versus unprimed letters does not matter. This can be guaranteed by choosing the following set of hypotheses:

$$\mathsf{pcp} = \{\ \mathtt{a} \cdot \mathtt{a}' = \mathtt{a}' \cdot \mathtt{a}, \quad \mathtt{a} \cdot \mathtt{b}' = \mathtt{b}' \cdot \mathtt{a}, \quad \mathtt{b} \cdot \mathtt{b}' = \mathtt{b}' \cdot \mathtt{b}, \quad \mathtt{b} \cdot \mathtt{a}' = \mathtt{a}' \cdot \mathtt{b}\ \}$$

Intuitively, we have a no-instance of PCP if and only if every possible arrangement of the words (encoded by e) represents two different words (as encoded by f). This is the key idea that allows one to show that $[\![e]\!] \subseteq [\![f]\!]^{\mathsf{pcp}}$ if and only if we have a no-instance of PCP.

Example 4.17. Suppose a concurrent programming language is implemented such that only single primitive actions could run in parallel, i.e., it is *synchronous* [Pri10]. Hence, given a program starting with the action $\mathtt{a}$, and another starting with the action $\mathtt{b}$, their parallel composition would first run $\mathtt{a}$ in parallel with $\mathtt{b}$ before continuing with the remaining parallel program. Furthermore, performing an action $\mathtt{a}$ in parallel with itself would be the same as simply running $\mathtt{a}$.

We could encode this type of behaviour using the following set of hypotheses

$$\mathsf{sync} = \big\{ \alpha \cdot e \parallel \beta \cdot f = (\alpha \parallel \beta) \cdot (e \parallel f) \ :\ \alpha, \beta \in \Sigma^\dagger,\ e, f \in \mathcal{T} \big\} \cup \{ \mathtt{a} \parallel \mathtt{a} = \mathtt{a} \ :\ \mathtt{a} \in \Sigma \}$$

where $\Sigma^\dagger = \{\mathtt{a}_1 \parallel \cdots \parallel \mathtt{a}_n \ :\ \mathtt{a}_1, \ldots, \mathtt{a}_n \in \Sigma\}$. It is not hard to show that $[\![\mathtt{a}^* \parallel \mathtt{a}^*]\!]^{\mathsf{sync}} = [\![\mathtt{a}^*]\!]^{\mathsf{sync}}$. On the other hand, $\mathtt{a}^* \parallel \mathtt{a}^* \equiv^{\mathsf{sync}} \mathtt{a}^*$ does *not* hold [WBK+19]. Hence, for the hypotheses in sync, the model constructed in the previous section identifies expressions that are not provably equal.

Remark 4.18. If we augment BKA congruence with *star-continuity*, i.e., the infinitary quasi-equation $e \cdot f^* \cdot g \equiv \sum_{n \in \mathbb{N}} e \cdot f^n \cdot g$ where $f^0 = 1$ and $f^{n+1} = f \cdot f^n$ [Con71], then for all $e, f \in \mathcal{T}$ without $\parallel$ (i.e., rational expressions), $[\![e]\!]^H = [\![f]\!]^H$ *does* imply $e \equiv^H f$, regardless of H [DKP+19].

Thus, the question becomes: how should H look to positively answer the questions about decidability and completeness? We formalise this as a property of hypotheses, as follows.

Definition 4.19 (Decidability, completeness)**.** Let H be a set of hypotheses, and let $e, f \in \mathcal{T}$. We call H *decidable* if $[\![e]\!]^H = [\![f]\!]^H$ is decidable. We call H *complete* if $[\![e]\!]^H = [\![f]\!]^H$ implies $e \equiv^H f$.

Example 4.20. The empty set of hypotheses is decidable and complete, as a result of Theorems 3.42 and 3.51; similarly, the set of hypotheses $\mathsf{all} = \{e = f \ : \ e, f \in \mathcal{T}\}$ is decidable and complete, as well. In contrast, we have seen that the set of hypotheses pcp from Example 4.16 is undecidable, and the set of hypotheses sync from Example 4.17 is incomplete.

We will tackle the question of completeness and decidability by relating sets of hypotheses, showing how, in the right circumstances, these properties can be carried over from one set of hypotheses to the next. To this end, we first need the notion of *implication* between hypotheses, which shows how one set of hypotheses can be "stronger" than another.

Definition 4.21 (Implication)**.** We say that H *implies* H' if we can use the hypotheses in H to prove those of H', i.e., if for every hypothesis $e \leq f \in H'$ it holds that $e \leqq^H f$.

Example 4.22. Every set of hypotheses implies the empty set of hypotheses; similarly, the set of hypotheses all from Example 4.20 implies every set of hypotheses. Less trivially, the set of hypotheses $\mathsf{comm} = \{\mathsf{a} \cdot \mathsf{b} = \mathsf{a} \parallel \mathsf{b} \ : \ \mathsf{a}, \mathsf{b} \in \Sigma\}$ implies pcp, since $\mathsf{a} \cdot \mathsf{b}' \equiv^{\mathsf{comm}} \mathsf{a} \parallel \mathsf{b}' \equiv \mathsf{b}' \parallel \mathsf{a} \equiv^{\mathsf{comm}} \mathsf{b}' \cdot \mathsf{a}$, and similarly for the other hypotheses in pcp. On the other hand, comm does not imply sync, because $\mathsf{a} \notin [\![\mathsf{a} \parallel \mathsf{a}]\!]^{\mathsf{comm}}$, meaning that $[\![\mathsf{a}]\!]^{\mathsf{comm}} \neq [\![\mathsf{a} \parallel \mathsf{a}]\!]^{\mathsf{comm}}$, and hence $\mathsf{a} \equiv^{\mathsf{comm}} \mathsf{a} \parallel \mathsf{a}$ does not hold.

By itself, implication is not very useful to show that decidability or completeness of one set of hypotheses can be transposed to another. This is witnessed by the fact that the decidable set of hypotheses all implies the undecidable set of hypotheses pcp, which in turn implies the decidable empty set of hypotheses, and similarly for sync and completeness. What we *can* do is show that mutual implication means that either both sets of hypotheses are decidable, or neither is, as follows.

Lemma 4.23. *Let H and H' be sets of hypotheses such that H implies H'.*

(i) If $e, f \in \mathcal{T}$ with $e \equiv^{H'} f$, then $e \equiv^H f$.

(ii) If $L \subseteq \mathsf{Pom}$, then $L^{H'} \subseteq L^H$.

(iii) If H' also implies H, then H is decidable (resp. complete) if and only if H' is, too.

We thus need something more asymmetrical, to get from a "complicated" set of hypotheses H to a "simpler" set of hypotheses H', where completeness or decidability might be easier to prove. Ultimately, we would like H' to be the empty set of hypotheses, where these questions are settled. The idea of a *reduction* is to show that we can shed hypotheses by massaging expressions: given an expression e, we obtain an expression e' that is equivalent to e (under H), and whose H'-closed semantics is isomorphic to the H-closed semantics of e. The ideas behind reductions are not new [Coh94; KS96; AFG$^+$14; KM14; LS17; DKP$^+$19]; what follows is merely a formalisation.

Definition 4.24 (Reduction)**.** Let H and H' be sets of hypotheses such that H implies H'. A computable function $r : \mathcal{T} \to \mathcal{T}$ is a *reduction* from H to H' when both of the following are true:

(i) for $e \in \mathcal{T}$, it holds that $e \equiv^H r(e)$, and

(ii) for $e, f \in \mathcal{T}$, if $[\![e]\!]^H = [\![f]\!]^H$, then $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$.

If the first requirement is replaced by the condition that $r(e) \leqq^H e$ and $e \leqq^{H'} r(e)$, and the latter requirement is replaced by the condition that for $e \in \mathcal{T}$ we have $[\![e]\!]^H = [\![r(e)]\!]^{H'}$, we say that r is *strong*. We call H *(strongly) reducible* to H' if there exists a (strong) reduction from H to H'.

Remark 4.25. Any strong reduction r is a reduction, since $e \leqq^{H'} r(e)$ means $e \leqq^H r(e)$, and hence $e \equiv^H r(e)$. Moreover, the second condition also holds, since $[\![r(e)]\!]^{H'} = [\![e]\!]^H = [\![f]\!]^H = [\![r(f)]\!]^{H'}$.

Furthermore, if we have established that r satisfies the first property of strong reduction, then for the second property we need only check that $[\![e]\!]^H \subseteq [\![r(e)]\!]^{H'}$. After all, the other inclusion can be derived by noting that $[\![r(e)]\!]^{H'} \subseteq [\![r(e)]\!]^H = [\![e]\!]^H$, by Lemma 4.23 and Theorem 4.14.

Example 4.26. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Let $H = \{\mathtt{a} \leq \mathtt{b}\}$. We can define for $e \in \mathcal{T}$ the expression $r(e) \in \mathcal{T}$, which is e but with every occurrence of $\mathtt{b}$ replaced by $\mathtt{a}+\mathtt{b}$. For instance, $r(\mathtt{a} \cdot \mathtt{b}^* \parallel \mathtt{c}) = \mathtt{a} \cdot (\mathtt{a} + \mathtt{b})^* \parallel \mathtt{c}$. An inductive argument on the structure of e shows that r strongly reduces H to $\emptyset$.

Reduction carries decidability and completeness from one set of hypotheses to another.

Lemma 4.27. *Suppose H is reducible to H'. If H' is decidable (resp. complete), then so is H.*

Proof. Let r be the reduction from H to H', and let $e, f \in \mathcal{T}$.

For completeness, suppose that $[\![e]\!]^H = [\![f]\!]^H$. We then know that $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$. Hence, by completeness of H', we have that $r(e) \equiv^{H'} r(f)$. Because H implies H', it follows that $r(e) \equiv^H r(f)$; for the same reason, $e \equiv^H r(e)$ and $f \equiv^H r(f)$. In total, this tells us that $e \equiv^H f$.

For decidability, first note that if $[\![e]\!]^H = [\![f]\!]^H$, then also $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$ by definition of reduction. Conversely, if $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$, then $([\![r(e)]\!]^{H'})^H = ([\![r(f)]\!]^{H'})^H$, which in turn allows us to derive that $[\![r(e)]\!]^H = [\![r(f)]\!]^H$ by Lemma 4.23(ii). Hence, by soundness of $[\![-]\!]^H$ w.r.t. $\equiv^H$ and the fact that $e \equiv^H r(e)$ as well as $f \equiv^H r(f)$, we can conclude that $[\![e]\!]^H = [\![f]\!]^H$. Therefore, we can decide whether $[\![e]\!]^H = [\![f]\!]^H$ by checking whether $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$. □

Finding a reduction is not always entirely trivial. The remainder of this chapter is devoted to two particular types of reduction that are comparatively easier to establish, and which will be used to derive results about equivalence of concurrent programs later on.

4.2.1 Reification

It can happen that the hypotheses in H impose an internal algebraic structure on the letters in Σ. To peel away this layer of axioms, we can try to reduce to expressions over a smaller alphabet, rendering the structure on the letters irrelevant. In a sense, this kind of reduction shows that the equivalences between letters from the hypotheses can already be guaranteed by normalisation.

Example 4.28. Let Σ be the set of group expressions over a (finite) alphabet Λ, that is, Σ consists of the expressions generated by the grammar $g, h ::= \mathtt{u} \mid \mathtt{a} \in \Lambda \mid g \circ h \mid \overline{g}$. Furthermore, let $\simeq$ be the smallest congruence on Σ generated by the group axioms, i.e., for all $g, h, i \in \Lambda$ it holds that

$$g \circ (h \circ i) \simeq (g \circ h) \circ i \qquad g \circ \mathtt{u} \simeq g \simeq \mathtt{u} \circ g \qquad \overline{g} \circ g \simeq \mathtt{u} \simeq g \circ \overline{g}$$

Let $\mathsf{group} = \{g = h \ : \ g \simeq h\}$.[1] Since every letter is uniquely represented by the minimal equivalent group expression, we can replace every group expression in e with its reduced form, obtaining a group-equivalent expression e'. For instance, if $\Lambda = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then we send $\mathtt{a} \circ \overline{\mathtt{a}} \parallel \mathtt{b} \circ \mathtt{c} \circ \overline{\mathtt{c}}$ to $\mathtt{u} \parallel \mathtt{b}$.

We fix a subalphabet $\Gamma \subseteq \Sigma$. When $r : \Sigma \to \mathcal{T}(\Gamma)$, we extend r to a map from $\mathcal{T}(\Sigma)$ to $\mathcal{T}(\Gamma)$, by inductively applying r to expressions. We formalise the idea of reducing by replacing letters.

Definition 4.29 (Reification)**.** Let $r : \Sigma \to \mathcal{T}(\Gamma)$ be computable. We call r a *reification* when

(i) For all $\mathtt{a} \in \Sigma$, it holds that $r(\mathtt{a}) \equiv^H \mathtt{a}$.

(ii) For all $e \leq f \in H$, it holds that $r(e) \leqq^{H'} r(f)$.

[1]Recall from Convention 4.3 that this a shorthand for $\{g \leq h \ : \ g \simeq h\} \cup \{h \leq g \ : \ g \simeq h\}$.

Example 4.30. Continuing Example 4.28, let r be the function that sends a group expression to its reduced form; this is computable. We claim that r is a reification from group to $\emptyset$. We already know that for a group expression g, $r(g) \simeq g$, and hence $r(g) \equiv^{\mathsf{group}} g$. Second, if $e \leq f \in \mathsf{group}$, then $e \simeq f$. Since reduced forms are unique, we have $r(e) = r(f)$, and hence $r(e) \leqq^{\emptyset} r(f)$.

To show that reification can give rise to reduction, we need some technical properties. First, note that a reification r associates with every $\mathtt{a} \in \Sigma$ a pomset language $[\![r(\mathtt{a})]\!]$. This means that we can view a reification r as a substitution $r : \Sigma \to 2^{\mathsf{SP}}$. We can then prove the following.

Lemma 4.31. *Let $r : \Sigma \to \mathcal{T}(\Gamma)$ be a reification.*

(i) For all $e \in \mathcal{T}$, it holds that $r([\![e]\!]) = [\![r(e)]\!]$.

(ii) For all $L \subseteq \mathsf{SP}(\Sigma)$, it holds that $r(L^H)^{H'} = r(L)^{H'}$.

Using the above and some earlier properties, we then arrive at the desired result.

Lemma 4.32. *Suppose H implies H'. Any reification from H to H' is a reduction from H to H'.*

Proof. Note that r is computable by definition of reification. The first condition, i.e., that for $e \in \mathcal{T}$ we have $r(e) \equiv^H e$, can be checked using the first property of reification by induction on e.

To prove the second condition, suppose that $[\![e]\!]^H = [\![f]\!]^H$. Using Lemma 4.31, we derive

$$[\![r(e)]\!]^{H'} = r([\![e]\!])^{H'} = r([\![e]\!]^{H})^{H'} = r([\![f]\!]^{H})^{H'} = r([\![f]\!])^{H'} = [\![r(f)]\!]^{H'} \qquad \square$$

Since group implies $\emptyset$, the fact that we have a reification from group to $\emptyset$ (c.f. Example 4.30) in combination with the above allows us to conclude that group is both decidable and complete.

4.2.2 Lifting

Hypotheses have already been studied extensively at the level of rational expressions. For instance, Cohen studied hypotheses of the forms $e = 0$ and $e \leq 1$ [Coh94], Kozen considered commutativity conditions of the form $\mathtt{a} \cdot \mathtt{b} = \mathtt{b} \cdot \mathtt{a}$ that we saw in Example 4.16 [Koz96], Kozen and Mamouras studied a wide class of hypotheses that could be presented as rewriting systems [KM14], and Doumane et al. considered (among other things) hypotheses of the form $1 = \mathtt{a}_1 + \cdots + \mathtt{a}_n$ [DKP+19].

Ideally, one would like to generalize those procedures to series-rational expressions without having to modify their internals. Similarly, when coming up with a new reduction, it is sometimes easier to focus on the fragment of expressions without parallel composition, and include this operator at a later stage. We shall derive a criterion that allows us to perform exactly such a lifting; the underlying idea is that the reduction must be one that is more or less agnostic of concurrency.

Example 4.33. The reductions in Example 4.26 and Example 4.28 were already correct for sr-expressions without parallel composition, and can then be extended by induction on the number of occurrences of $\parallel$, defining the reduction of $e \parallel f$ to be reductions of e and f, composed in parallel.

In contrast, consider $H = \{\mathsf{a} \leq 1\}$ for some $\mathsf{a} \in \Sigma$. Although H can be reduced to $\emptyset$ for rational expressions [Coh94], it is not obvious how this would work for pomset languages. In particular, if $1 \in L$, then $1 \parallel \cdots \parallel 1 \in L$ for any number of 1's, and hence $\mathsf{a} \parallel \cdots \parallel \mathsf{a} \in L^H$ for any number of a's. This precludes the possibility of a strong reduction to $\emptyset$, because it shows that $[\![1]\!]^H$ is a pomset language of unbounded (parallel) width, which cannot be expressed by any $e \in \mathcal{T}$ [LW00].

We start by formalising the idea of a reduction that applies only to sr-expressions without $\parallel$. To this end, we should first properly define rational expressions and their equivalence.

Definition 4.34 (Rational expressions)**.** The set of *rational expressions*, denoted $\mathcal{T}_\mathsf{R}$, consists of the expressions in $\mathcal{T}$ that do not contain $\parallel$; equivalently, it is generated by

$$e, f ::= 0 \mid 1 \mid \mathsf{a} \in \Sigma \mid e + f \mid e \cdot f \mid e^*$$

We say that $\approx$ is a *Kleene algebra congruence*, or *KA congruence* for short, if $\approx$ is a congruence that satisfies all the axioms of BKA congruence that do not involve $\parallel$. We write $\equiv_\mathsf{R}$ for the smallest KA congruence, and use $e \leqq_\mathsf{R} f$ as a shorthand for $e + f \equiv_\mathsf{R} f$. The set $\mathcal{F}_\mathsf{R}$ of *accepting rational expressions* is defined similarly to $\mathcal{F}$, but without parallel composition; equivalently, $\mathcal{F}_\mathsf{R} = \mathcal{F} \cap \mathcal{T}_\mathsf{R}$.

It should be clear that the axioms of KA congruence are precisely the well-known axioms of Kleene algebra [Koz94]; the semantics of rational expressions given by $[\![-]\!]$ is exactly the well-known semantics in terms of languages, where words are represented by totally ordered pomsets.

We can then define the non-parallel analogue of hypotheses and closure. Because the semantics of rational expressions includes totally ordered pomsets exclusively, we may forego the use of pomset contexts and use what is essentially the closure operator from [DKP$^+$19].

Definition 4.35 (Sequential hypotheses)**.** A *sequential hypothesis* is a hypothesis $e \leq f$ where $e, f \in \mathcal{T}_\mathsf{R}$. The relation $\equiv^H_\mathsf{R}$ is generated from a set of sequential hypotheses H and $\equiv_\mathsf{R}$ as before.

Given a set of hypotheses H and a language $L \subseteq \Sigma^*$, the *sequential closure* of L w.r.t. H, written $L^{\langle H \rangle}$, is the least language that contains L and satisfies the following inference rule

$$\frac{e \leq f \in H \qquad w, x \in \Sigma^* \qquad w \cdot [\![f]\!] \cdot x \subseteq L^{\langle H \rangle}}{w \cdot [\![e]\!] \cdot x \subseteq L^{\langle H \rangle}}$$

Next, we define what it means to have a reduction that applies to rational expressions exclusively, using rational expressions, equivalence between rational expressions, and the above notion of closure.

Definition 4.36 (Sequential reduction)**.** Suppose that H implies H'. A computable function $r : \mathcal{T}_\mathsf{R} \to \mathcal{T}_\mathsf{R}$ is a *sequential reduction* from H to H' when the following hold:

(i) for $e \in \mathcal{T}_\mathsf{R}$, it holds that $e \leqq^{H'}_\mathsf{R} r(e)$ and $r(e) \leqq^{H}_\mathsf{R} e$, and

(ii) for $e \in \mathcal{T}_\mathsf{R}$, it holds that $[\![e]\!]^{\langle H \rangle} = [\![r(e)]\!]^{\langle H' \rangle}$.

H *sequentially reduces* to H' if there exists a sequential reduction from H to H'.

To recover a strong reduction from a sequential reduction, we can try to simply extend it homomorphically, setting $r(e \parallel f) = r(e) \parallel r(f)$. However, this does not always work.

Example 4.37. Let $H = \{\mathtt{a} \leq \mathtt{b} + \mathtt{c}\}$. It is possible to find a sequential reduction r from H to the empty set of hypotheses. If we try to lift r to sr-expressions as outlined above, we find that, since $[\![\mathtt{b} + \mathtt{c}]\!] \subseteq [\![(\mathtt{b} + 1) \parallel (\mathtt{c} + 1)]\!]$ and $\mathtt{a} \leq \mathtt{b} + \mathtt{c}$, also $\mathtt{a} \in [\![\mathtt{a}]\!] \subseteq [\![(\mathtt{b} + 1) \parallel (\mathtt{c} + 1)]\!]^H$. On the other hand,

$$[\![r(\mathtt{b} + 1) \parallel r(\mathtt{c} + 1)]\!]^{\emptyset} = [\![r(\mathtt{b} + 1)]\!] \parallel [\![r(\mathtt{c} + 1)]\!] = [\![\mathtt{b} + 1]\!]^{H} \parallel [\![\mathtt{c} + 1]\!]^{H} = [\![(\mathtt{b} + 1) \parallel (\mathtt{c} + 1)]\!]$$

which does not contain $\mathtt{a}$. Note that this argument applies to any sequential reduction r.

From the above, we learn that a hypothesis $e \leq f$ where the semantics of f contains more than one word might be problematic. In Example 4.33, we have also seen that having the semantics of f contain the empty pomset may also give us trouble. Excluding both of these cases then naturally leads to the following restriction on the right-hand side of hypotheses.

Definition 4.38 (Grounded hypotheses)**.** A sequential hypothesis $e \leq f$ is *grounded* if $f = \mathtt{a}_1 \cdots \mathtt{a}_n$ for some $\mathtt{a}_1, \dots, \mathtt{a}_n \in \Sigma$ with $n \geq 1$. A set of hypotheses H is grounded if its members are.

Example 4.39. The hypotheses in $\mathtt{pcp}$ (c.f. Example 4.16) are grounded. On the other hand, the hypothesis $\mathtt{a} \leq 1$ discussed in Example 4.33 is not grounded, because 1 is not a sequence of letters. Similarly, neither is the hypothesis $\mathtt{a} \leq \mathtt{b} + \mathtt{c}$ discussed in Example 4.37.

Grounded hypotheses are sufficiently well-behaved to establish the following technical properties.

Lemma 4.40. *Let H be grounded. If $L \subseteq \Sigma^*$, then $L^H = L^{\langle H \rangle}$.*

Furthermore, for $L, L' \subseteq \mathsf{SP}$, we have that $(L \parallel L')^H = L^H \parallel L'^H$.

We can then state the desired result, which says that sequential reductions can be turned into strong reductions on series-rational expressions, provided the hypotheses are grounded.

Lemma 4.41. *Let H and H' be grounded, and let r be a sequential reduction from H to H'. If we extend r to $r : \mathcal{T} \to \mathcal{T}$ by setting $r(e \parallel f) = r(e) \parallel r(f)$, then r is a strong reduction from H to H'.*

Proof. We already know that H implies H'. Let r be the sequential reduction from H to H'. We extend r to a function $\mathcal{T} \to \mathcal{T}$ by acting homomorphically, i.e., $r(e \parallel f) = r(e) \parallel r(f)$; if r is computable, then this extension is, too. It is not hard to show that $e \leqq^{H'} r(e)$ and $r(e) \leqq^{H} e$.

For the last requirement, the proof proceeds by induction on the number of occurrences of $\parallel$ in e. In the base, where $\parallel$ does not occur, we have that $e \in \mathcal{T}_{\mathsf{R}}$. We can then derive by Lemma 4.40 that

$$[\![e]\!]^{H} = [\![e]\!]^{\langle H \rangle} = [\![r(e)]\!]^{\langle H' \rangle} = [\![r(e)]\!]^{H'}$$

For the inductive step, we have $e = e_0 \parallel e_1$. We then derive, using Lemma 4.40 and induction that

$$[\![e]\!]^{H} = [\![e_0]\!]^{H} \parallel [\![e_1]\!]^{H} = [\![r(e_0)]\!]^{H'} \parallel [\![r(e_1)]\!]^{H'} = [\![r(e)]\!]^{H'} \qquad \square$$

Thus, when H is grounded, we can find a strong reduction from H to H' by constructing a sequential reduction from H to H'. To find such a sequential reduction, we can specialise the results about series-rational systems to rational expressions, and adapt the relevant proofs [Bac75; Koz94].

Definition 4.42 (Rational systems)**.** Let Q be a finite set. A *rational system* on Q is a tuple $\mathcal{S} = \langle M, b \rangle$, where $M : Q^2 \to \mathcal{T}_{\mathsf{R}}$ and $b : Q \to \mathcal{T}_{\mathsf{R}}$. Let $\approx$ be a KA congruence on $\mathcal{T}_{\mathsf{R}}(\Delta)$ with $\Sigma \subseteq \Delta$, and let $e \in \mathcal{T}_{\mathsf{R}}$. We call $s : Q \to \mathcal{T}_{\mathsf{R}}(\Delta)$ a *$\langle \approx, e \rangle$-solution* to $\mathcal{S}$ if for every $q \in Q$ we have:

$$b(q) \cdot e + \sum_{q' \in Q} M(q, q') \cdot s(q') \lessapprox s(q)$$

Lastly, s is the *least* $\langle \approx, e \rangle$-solution if, for every such solution s' and every $q \in Q$ we have $s(q) \lessapprox s'(q)$.

Theorem 4.43. *Let $\mathcal{S} = \langle M, b \rangle$ be a rational system on Q. We can construct an $s : Q \to \mathcal{T}_{\mathsf{R}}$ such that, for any KA congruence $\approx$ on $\mathcal{T}_{\mathsf{R}}(\Delta)$ with $\Sigma \subseteq \Delta$ and any $e \in \mathcal{T}_{\mathsf{R}}$, the Q-vector $s^e : Q \to \mathcal{T}_{\mathsf{R}}$ given by $s^e(q) = s(q) \cdot e$ is the least $\langle \approx, e \rangle$-solution to $\mathcal{S}$.*

Convention 4.44. Keeping with previous nomenclature, we refer to the function $s : Q \to \mathcal{T}_{\mathsf{R}}$ obtained for any rational system $\mathcal{S}$ as the *least solution* to $\mathcal{S}$.

Remark 4.45. We should be careful to emphasize that Theorem 4.43 is not a special case of Theorem 3.60, because solutions to rational systems are defined in terms of KA congruence, and are required to contain rational expressions. On the other hand, the proof of Theorem 3.60 does not use parallel composition anywhere, so it can straightforwardly be turned into a proof of Theorem 4.43; conversely, any proof of Theorem 4.43 can be turned into a proof of Theorem 3.60.

4.2.3 Decomposition

When designing a reduction from H_0 to H_1, a compositional approach can be useful, because it can help manage complexity, and opens up the possibility of reusing reductions derived elsewhere. The most obvious way to do this is by first reducing H_0 to some intermediate set of hypotheses H', which is in turn reduced to H_1; the reductions involved can then be composed, as follows.

Lemma 4.46. *Let H_0, H_1 and H' be sets of hypotheses. If r is a reduction from H_0 to H', and r' is a reduction from H' to H_1, then $r' \circ r$ is a reduction from H_0 to H_1.*

Alternatively, we can find reductions by *decomposition* [DKP+19]. The idea here is to find several weaker sets of hypotheses, and show that closure w.r.t. H can be *factorised* into closure w.r.t. these sets of hypotheses, in some order. First, let us formalise factorisation.

Definition 4.47 (Factorisation)**.** H *factorises* into $H_1, \dots, H_n$ if for every $L \subseteq \mathsf{SP}$ we have

$$L^H = \left((L^{H_1}) \cdots\right)^{H_n}$$

Example 4.48 [DKP+19, Proposition 3]**.** Let $H = \{\mathtt{a}_1 + \cdots + \mathtt{a}_n = 1\}$ for $\mathtt{a}_1, \dots, \mathtt{a}_n \in \Sigma$. For rational languages, we can factorise H into two sets of hypotheses, $H^{\leq}, H^{\geq}$, where

$$H^{\leq} = \{\mathtt{a}_i \leq 1 \ : \ 1 \leq i \leq n\} \qquad\qquad H^{\geq} = \{1 \leq \mathtt{a}_1 + \cdots + \mathtt{a}_n\}$$

This can be generalised to the case where H consists of several hypotheses of the form $\mathtt{a}_1 + \cdots + \mathtt{a}_n = 1$. With a little more work, one can show that the same holds for series-rational languages.

If we can show that each of the sets of hypotheses that H factorises into strongly reduces to the last, we can conclude that H strongly reduces to this last set. For instance, if both $H^{\leq}$ and $H^{\geq}$ strongly reduce to $\emptyset$, then H strongly reduces to $\emptyset$ [DKP+19, Propositions 5 and 6].

Lemma 4.49. *Let H be a set of hypotheses that factorises into $H_1, \dots, H_n$, where for $1 \leq i \leq n$, H implies H_i, which strongly reduces to H_n. Then H strongly reduces to H_n.*

Proof. For $1 \leq i \leq n$, let r_i be the reduction from H_i to H_n. We choose $r'_m = r_m \circ \cdots \circ r_1$ for $0 \leq m \leq n$. We claim that $r = r'_n$ is a strong reduction from H to H_n; this function is computable.

First, note that H already implies H_n. It remains to verify the two conditions for strong reduction. Observe that for $1 \leq m \leq n$ we have $r'_m(e) = r_m(r'_{m-1}(e)) \leqq^{H_m} r'_{m-1}(e)$ and similarly $r'_{m-1}(e) \leqq^{H_n} r_m(r'_{m-1}(e)) = r'_m(e)$. Because H implies each H_i, we can lift the former to $\leqq^H$ (by Lemma 4.23); hence, $r'_m(e) \leqq^H r'_{m-1}(e)$ and $r'_{m-1}(e) \leqq^{H_n} r'_m(e)$ for $1 \leq m \leq n$; chaining these, we find that $r(e) = r'_n(e) \leqq^H r'_0(e) = e$ and similarly $e = r'_0(e) \leqq^{H_n} r'_n(e) = r(e)$.

For the second condition, first note that $[\![e]\!]^H = [\![r(e)]\!]^H$ by soundness; hence, $[\![r(e)]\!]^{H_n} \subseteq [\![e]\!]^H$ by Lemma 4.23. For the other inclusion, we first show that for $0 \leq m \leq n$, we have $(([\![e]\!])^{H_1} \cdots)^{H_m} \subseteq [\![r'_m(e)]\!]^{H_n}$, by induction on m. In the base, where $m = 0$, the claim holds trivially. For the inductive step, let $m > 0$ and note that $(([\![e]\!])^{H_1} \cdots)^{H_m} \subseteq ([\![r'_{m-1}(e)]\!]^{H_n})^{H_m}$. By Lemma 4.23, the latter is contained in $[\![r'_{m-1}(e)]\!]^{H_m}$. Because r_m is a strong reduction, this in turn is equal to $[\![r_m(r'_{m-1}(e))]\!]^{H_n} = [\![r'_m(e)]\!]^{H_n}$. Thus, by Lemma 4.11, we can conclude that $[\![e]\!]^H = ((([\![e]\!])^{H_1} \cdots)^{H_n})^{H_n} \subseteq [\![r'_n(e)]\!]^{H_n} = [\![r(e)]\!]^{H_n}$. □

We conclude this chapter with a useful result about factorising a set of hypotheses into a partition of those hypotheses, i.e., factorising $H = H_1 \cup \cdots \cup H_n$ into $H_1, \ldots, H_n$. Proving such a factorisation result is a very tedious exercise indeed. However, if we can show that each (ordered) pair of these smaller sets of hypotheses can be factorised, this leads to a factorisation of H.

Lemma 4.50. *Let $H_1, \ldots, H_n$ be sets of hypotheses. If, for $1 \leq i \leq j \leq n$, we have that $H_i \cup H_j$ factorises into H_i, H_j, then $H_1 \cup \cdots \cup H_n$ factorises into $H_1, \ldots, H_n$.*

Proof. Let $H = H_1 \cup \cdots \cup H_n$. We should prove that $L^H = (L^{H_1} \cdots)^{H_n}$.

For the inclusion from right to left, we proceed by induction on n. In the base, where $n = 0$, the claim holds immediately, since $L^\emptyset = L$. For the inductive step, assume the claim holds for n, i.e., that $((L^{H_1}) \cdots)^{H_n} \subseteq L^{H_1 \cup \cdots \cup H_n}$. We then find, using Lemma 4.23(ii) and Lemma 4.11, that

$$\left(\left((L^{H_1}) \cdots\right)^{H_n}\right)^{H_{n+1}} \subseteq \left(L^{H_1 \cup \cdots \cup H_n}\right)^{H_{n+1}} \subseteq \left(L^H\right)^H = L^H$$

For the other inclusion, we show that if $A \subseteq L^H$, then $A \subseteq ((L^{H_1}) \cdots)^{H_n}$, by induction on the construction of the former. In the base, where $A = L$, the claim holds immediately. In the inductive step, we obtain $e \leq f \in H_i$ for some $1 \leq i \leq n$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ such that $A = C[[\![e]\!]]$ and $C[[\![f]\!]] \subseteq L^H$. By induction, we then have that $C[[\![f]\!]] \subseteq ((L^{H_1}) \cdots)^{H_n}$. By the premise, we have for $1 \leq i \leq j \leq n$ and $K \subseteq \mathsf{SP}$ that $(K^{H_j})^{H_i} \subseteq K^{H_i \cup H_j} \subseteq (K^{H_i})^{H_j}$; thus, we conclude that

$$A \subseteq \left(\left((L^{H_1}) \cdots\right)^{H_n}\right)^{H_i} \subseteq \left(\left(\left((L^{H_1}) \cdots\right)^{H_i}\right)^{H_i} \cdots\right)^{H_n} \subseteq \left((L^{H_1}) \cdots\right)^{H_n}$$ □

Summary of this chapter The axioms of bi-Kleene algebra can be used to reason about general program equivalence, but further specialisation is required when reasoning about particular programs. Hypotheses provide a tool for such specialisation. We formalised existing methods to achieve results about completeness and decidability using reductions, and proposed several tools for deriving these reductions. We also formalised an existing method to compose several existing reductions in non-trivial ways, thereby deriving new reductions for more complicated sets of hypotheses.

4.A Proofs about closure

Lemma 4.11 (c.f. [DKP+19, Lemma 1])**.** *Let $L, K \subseteq \mathsf{Pom}$. Then $L \subseteq K^H$ if and only if $L^H \subseteq K^H$.*

Proof. First, suppose $L \subseteq K^H$. We show that for all $A \subseteq L^H$, we have that $A \subseteq K^H$, by induction on the construction of $A \subseteq L^H$. In the base, where $A = L$, we have $A \subseteq K^H$ by the premise. If $A = C[\llbracket e \rrbracket]$ with $e \leq f \in H$ and $C[\llbracket f \rrbracket] \subseteq L^H$, then by induction $C[\llbracket f \rrbracket] \subseteq K^H$, and thus $A = C[\llbracket e \rrbracket] \subseteq K^H$. The other implication is trivial, since $L \subseteq L^H \subseteq K^H$. □

Lemma 4.13 (c.f. [DKP+19, Lemma 2])**.** *Let $L, K \subseteq \mathsf{Pom}$. The following hold:*

$$(L \cup K)^H = \left(L^H \cup K^H\right)^H \qquad (L \cdot K)^H = \left(L^H \cdot K^H\right)^H$$

$$(L \parallel K)^H = \left(L^H \parallel K^H\right)^H \qquad (L^*)^H = \left(\left(L^H\right)^*\right)^H$$

Proof. First, recall that closure is necessarily monotone: if $L \subseteq K$, then $L^H \subseteq K^H$. After all, we can find that $L \subseteq K \subseteq K^H$, whence $L^H \subseteq K^H$ by Lemma 4.11. In all equalities, the inclusion from left to right is a consequence of monotonicity. For instance, since $L \subseteq L^H$ and $K \subseteq K^H$, we have that $L \cup K \subseteq L^H \cup K^H$, and hence $(L \cup K)^H \subseteq (L^H \cup K^H)^H$, and similarly for the other claims.

To show that $(L^H \cup K^H)^H \subseteq (L \cup K)^H$, observe that $L^H, K^H \subseteq (L \cup K)^H$ by monotonicity, and hence $L^H \cup K^H \subseteq (L \cup K)^H$. We then conclude by Lemma 4.11 that $(L^H \cup K^H)^H \subseteq (L \cup K)^H$.

To show that $(L^H \cdot K^H)^H \subseteq (L \cdot K)^H$ and $(L^H \parallel K^H)^H \subseteq (L \parallel K)^H$, it suffices to prove that $L^H \cdot K^H \subseteq (L \cdot K)^H$ and $L^H \parallel K^H \subseteq (L \parallel K)^H$. To this end, we first prove the following.

Fact 4.A.1. *If $A \subseteq L^H$ and $C \in \mathsf{PC}^{\mathsf{sp}}$, then $C[A] \subseteq C[L]^H$.*

Proof. We proceed by induction on the construction of $A \subseteq L^H$. In the base, where $A \subseteq L^H$ because $A = L$, the claim holds immediately. For the inductive step, if $A \subseteq L^H$ because there exist $C' \in \mathsf{PC}^{\mathsf{sp}}$ and $e \leq f$ such that $C'[\llbracket f \rrbracket] \subseteq L^H$ and $A = C'[\llbracket e \rrbracket]$, then choose $C'' = C[C']$. Since $C''[\llbracket f \rrbracket] = C[C'[\llbracket f \rrbracket]] \subseteq C[L]^H$, by induction, also $C[A] = C[C'[\llbracket e \rrbracket]] = C''[\llbracket e \rrbracket] \subseteq C[L]^H$. □

The above straightforwardly gives rise to the following containments for all $L, K \subseteq \mathsf{Pom}$:

$$L \cdot K^H \subseteq (L \cdot K)^H \qquad L^H \cdot K \subseteq (L \cdot K)^H \qquad L \parallel K^H \subseteq (K \parallel K)^H$$

The desired inclusions then follow; after all, we can derive that

$$L^H \cdot K^H \subseteq \left(L \cdot K^H\right)^H \subseteq \left((L \cdot K)^H\right)^H \subseteq (L \cdot K)^H$$

$$L^H \parallel K^H \subseteq \left(L \parallel K^H\right)^H \subseteq \left((L \parallel K)^H\right)^H \subseteq (L \parallel K)^H$$

where the last inclusion follows from Lemma 4.11.

To show that $((L^H)^*)^H \subseteq (L^*)^H$, it suffices to prove that $(L^H)^* \subseteq (L^*)^H$. To this end, we first argue that for all $n \in \mathbb{N}$, it holds that $(L^H)^n \subseteq (L^*)^H$, by induction on n. In the base, where $n = 0$, we have that $(L^H)^0 = \{1\} \subseteq (\{1\})^H \subseteq (L^*)^H$ by monotonicity. For the inductive step, suppose the claim holds for n. We then calculate, using Lemma 4.11 and the second equality, that

$$(L^H)^{n+1} = (L^H)^n \cdot L^H \subseteq (L^*)^H \cdot L^H \subseteq (L^* \cdot L)^H \subseteq (L^*)^H$$

Putting this together, we have that

$$(L^H)^* = \bigcup_{n\in\mathbb{N}} (L^H)^n \subseteq \bigcup_{n\in\mathbb{N}} (L^*)^H \subseteq (L^*)^H \qquad \square$$

Theorem 4.14 (c.f. [DKP+19, Theorem 2])**.** *If $e \equiv^H f$, then $[\![e]\!]^H = [\![f]\!]^H$.*

Proof. We proceed by induction on the construction of $\equiv^H$. In the base, there are three cases. First, if $e \equiv^H f$ because $e = f$, then the claim holds immediately. Otherwise, if $e \equiv^H f$ because $e \equiv f$, then $[\![e]\!] = [\![f]\!]$ by Theorem 3.47, and therefore $[\![e]\!]^H = [\![f]\!]^H$. Lastly, if $e \leqq^H f$ because $e \leq f \in H$, then by definition of closure we have $[\![e]\!] \subseteq [\![f]\!]^H$, and hence $[\![e]\!]^H \subseteq [\![f]\!]^H$ by Lemma 4.11.

For the inductive step, there are several cases to consider. If $e \equiv^H f$ because of a congruence rule, then for instance $e = e_0 + e_1$ and $f = f_0 + f_1$ with $e_i \equiv^H f_i$ for $i \in \{0, 1\}$. Thus, by induction, we have that $[\![e_i]\!]^H = [\![f_i]\!]^H$ for $i \in \{0, 1\}$. Using Lemma 4.13, we then derive that

$$\begin{aligned} [\![e_0 + e_1]\!]^H = ([\![e_0]\!] \cup [\![e_1]\!])^H &= \left([\![e_0]\!]^H \cup [\![e_1]\!]^H\right)^H \\ &= \left([\![f_0]\!]^H \cup [\![f_1]\!]^H\right)^H = ([\![f_0]\!] \cup [\![f_1]\!])^H = [\![f_0 + f_1]\!]^H \end{aligned}$$

The other cases for congruence are argued similarly.

Otherwise, if $e \equiv^H f$ because $f \equiv^H e$, then by induction we have that $[\![f]\!]^H = [\![e]\!]^H$, and so the claim follows. Lastly, if $e \equiv^H f$ because there exists a $g \in \mathcal{T}$ with $e \equiv^H g$ and $g \equiv^H f$, then $[\![e]\!]^H = [\![g]\!]^H$ and $[\![g]\!]^H = [\![f]\!]^H$ by induction, and hence the claim follows again. $\square$

4.B Proofs about reductions

Lemma 4.23. *Let H and H' be sets of hypotheses such that H implies H'.*

(i) If $e, f \in \mathcal{T}$ with $e \equiv^{H'} f$, then $e \equiv^H f$.

(ii) If $L \subseteq \mathsf{Pom}$, then $L^{H'} \subseteq L^H$.

(iii) If H' also implies H, then H is decidable (resp. complete) if and only if H' is, too.

Proof. We treat the claims in the order given.

(i) By induction on $e \equiv^{H'} f$. In the base, we have two cases. On the one hand, if $e \equiv^{H'} f$ because $e \equiv f$, then $e \equiv^{H} f$ immediately. On the other hand, if $e \leqq^{H'} f$ because $e \leq f \in H'$, then $e \leqq^{H} f$ by the premise. For the inductive step, there are again two cases to consider.

- The proof for congruence is straightforward. For instance, if $e = e_0 + e_1$ and $f = f_0 + f_1$ with $e_i \equiv^{H'} f_i$ for $i \in \{0,1\}$, then by induction $e_i \equiv^{H} f_i$ for $i \in \{0,1\}$, hence $e \equiv^{H} f$.
- If $e \leqq^{H'} f$ because of a fixpoint rule, then we proceed as follows. First, if $e = g \cdot h^*$ and $g + h \cdot f \leqq^{H'} f$, then by induction $g + h \cdot f \leqq^{H} f$. We then conclude that $e = g \cdot h^* \leqq^{H} f$ as well. The proof for the other fixpoint rule is similar.

(ii) We show that if $A \subseteq L^{H'}$, then $A \subseteq L^{H}$, by induction on the construction of $A \subseteq L^{H'}$. In the base, where $A = L$, the claim holds by definition. For the inductive step, there exist $C \in \mathsf{PC}^{\mathsf{sp}}$ and $e \leq f \in H'$ with $A = C[\llbracket e \rrbracket]$ and $C[\llbracket f \rrbracket] \subseteq L^{H'}$; thus $C[\llbracket f \rrbracket] \subseteq L^{H}$ by induction. By the premise we furthermore know that $e \leqq^{H} f$, and hence $\llbracket e \rrbracket^{H} \subseteq \llbracket f \rrbracket^{H}$ by soundness. By Lemma 4.13, we derive that $C[\llbracket e \rrbracket]^{H} \subseteq C[\llbracket f \rrbracket]^{H} \subseteq L^{H}$ and hence $C[\llbracket e \rrbracket] \subseteq L^{H}$ by Lemma 4.11.

(iii) For decidability, note that by the previous property we have for $e, f \in \mathcal{T}$ that $\llbracket e \rrbracket^{H} = \llbracket f \rrbracket^{H}$ holds if and only if $\llbracket e \rrbracket^{H'} = \llbracket f \rrbracket^{H'}$. Hence we can decide one by checking the other.

For completeness, suppose that H is complete. To argue completeness of H', let $e, f \in \mathcal{T}$ with $\llbracket e \rrbracket^{H'} = \llbracket f \rrbracket^{H'}$. By (ii), we have that $\llbracket e \rrbracket^{H} = \llbracket f \rrbracket^{H}$; by completeness of H this implies that $e \equiv^{H} f$, and by (i) and the fact that H' implies H, we have that $e \equiv^{H'} f$. The proof that completeness of H' yields completeness of H is similar. □

To prove Lemma 4.31, we need the following technical lemma.

Lemma 4.B.1. *Let $r : \Sigma \to 2^{\mathsf{SP}}$ be a substitution, and suppose we extend this substitution to $r : \Sigma \cup \{\square\} \to 2^{\mathsf{SP}(\Sigma \cup \{\square\})}$ by setting $r(\square) = \{\square\}$. In that case, it holds for for all $L \subseteq \mathsf{SP}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ that $r(C) \subseteq \mathsf{PC}^{\mathsf{sp}}$ as well as $r(C[L]) = \bigcup_{D \in r(C)} D[r(L)]$.*

Proof. The proof of the first claim proceeds by induction on the construction of C. In the base, $C = \square$, in which case $r(C) = \{\square\} \subseteq \mathsf{PC}^{\mathsf{sp}}$. For the inductive step, there are three cases. If $C = C' \cdot V$, then $r(C) = r(C') \cdot r(V)$. By induction, we know that $r(C') \subseteq \mathsf{PC}^{\mathsf{sp}}$; since $r(V) \subseteq \mathsf{SP}$, the claim then follows by definition of $\mathsf{PC}^{\mathsf{sp}}$. The other cases are similar.

The proof of the second claim proceeds by induction on the construction of C. In the base, $C = \square$, in which case $r(C) = \{\square\}$, and hence $r(C[L]) = r(L) = \bigcup_{D \in r(C)} D[r(L)]$. For the inductive step, there are three cases. If $C = C' \cdot V$, then $r(C) = r(C') \cdot r(V)$. We observe:

- If $D' \in r(C')$ and $U \in D'[r(L)] \cdot r(V)$, then there exists a $D \in r(C)$ such that $U \in D[r(L)]$. To see this, note that $U = D'[W] \cdot X$ for $W \in r(L)$ and $X \in r(V)$. If we then choose $D = D' \cdot X \in r(C') \cdot r(V) = r(C)$, we find that $U \in D'[r(L)] \cdot X = D[r(L)]$.
- If $D \in r(C)$, then there exists a $D' \in r(C')$ with $D[r(L)] \subseteq D'[r(L)] \cdot r(V)$. To see this, note that $D = D' \cdot W$ for $D' \in r(C')$ and $W \in r(V)$, and $D[r(L)] = D'[r(L)] \cdot W \subseteq D'[r(L)] \cdot r(V)$.

Hence, we derive that

$$\begin{aligned}
r(C[L]) &= r(C'[L] \cdot V) && \text{(Def. } C[-]) \\
&= r(C'[L]) \cdot r(V) && \text{(Def. } r \text{ on languages)} \\
&= \left(\bigcup\nolimits_{D' \in r(C')} D'[r(L)]\right) \cdot r(V) && \text{(Induction)} \\
&= \bigcup_{D' \in r(C')} D'[r(L)] \cdot r(V) && \text{(Distributivity)} \\
&= \bigcup_{D \in r(C)} D[r(L)] && \text{(Observations above)}
\end{aligned}$$

The other cases can be derived similarly. □

Lemma 4.31. *Let $r : \Sigma \to \mathcal{T}(\Gamma)$ be a reification.*

(i) For all $e \in \mathcal{T}$, it holds that $r(\llbracket e \rrbracket) = \llbracket r(e) \rrbracket$.

(ii) For all $L \subseteq \mathsf{SP}(\Sigma)$, it holds that $r(L^H)^{H'} = r(L)^{H'}$.

Proof. We treat the claims in the order given.

(i) The proof proceeds by induction on the construction of e. In the base, there are two cases to consider. First, if $e = 0$ or $e = 1$, then $r(\llbracket e \rrbracket) = \llbracket e \rrbracket = \llbracket r(e) \rrbracket$. Otherwise, if $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $r(\llbracket e \rrbracket) = r(\{\mathtt{a}\}) = r(\mathtt{a}) = \llbracket r(\mathtt{a}) \rrbracket$.

For the inductive step, the proof is straightforward. For instance, when $e = e_0 + e_1$, we derive

$$\begin{aligned}
r(\llbracket e_0 + e_1 \rrbracket) &= r(\llbracket e_0 \rrbracket \cup \llbracket e_1 \rrbracket) && \text{(Def. } \llbracket - \rrbracket) \\
&= r(\llbracket e_0 \rrbracket) \cup r(\llbracket e_1 \rrbracket) && \text{(Def. } r \text{ on languages)} \\
&= \llbracket r(e_0) \rrbracket \cup \llbracket r(e_1) \rrbracket && \text{(Induction)} \\
&= \llbracket r(e_0) + r(e_1) \rrbracket && \text{(Def. } \llbracket - \rrbracket) \\
&= \llbracket r(e_0 + e_1) \rrbracket && \text{(Def. } r \text{ on expressions)}
\end{aligned}$$

The other cases can be shown similarly.

(ii) For the inclusion from right to left, note that $L \subseteq L^H$, and hence $r(L) \subseteq r(L^H)$, which in turn means that $r(L)^{H'} \subseteq r(L^H)^{H'}$. By Lemma 4.11, the other inclusion can be proved by showing that $r(L^H) \subseteq r(L)^{H'}$. As usual for such statements, we proceed by induction on the construction of L^H, showing that for all $A \subseteq L^H$ it holds that $r(A) \subseteq r(L)^{H'}$. In the base, where $A = L$, we have $r(L) \subseteq r(L)^{H'}$ by definition of closure.

For the inductive case, assume $A = C[\llbracket e \rrbracket]$, for some C and $e \leq f \in H'$ such that $C[\llbracket f \rrbracket] \subseteq L^H$. By induction, we have that $r\left(C[\llbracket f \rrbracket]\right) \subseteq r(L)^{H'}$. Furthermore, since $e \leq f \in H$, by definition of a reification we have $r(e) \leqq^{H'} r(f)$, so by soundness $\llbracket r(e) \rrbracket^{H'} \subseteq \llbracket r(f) \rrbracket^{H'}$. By Fact 4.A.1, $D\left[\llbracket r(e) \rrbracket\right]^{H'} \subseteq D\left[\llbracket r(f) \rrbracket\right]^{H'}$ for $D \in \mathsf{PC}^{\mathsf{sp}}$. Using this observation, we may then derive:

$$\begin{aligned}
r(C[\llbracket e \rrbracket]) &= \bigcup_{D \in r(C)} D[r(\llbracket e \rrbracket)] && \text{(Lemma 4.B.1)} \\
&\subseteq \bigcup_{D \in r(C)} D[\llbracket r(e) \rrbracket]^{H'} && \text{(Def. closure)} \\
&\subseteq \bigcup_{D \in r(C)} D[\llbracket r(f) \rrbracket]^{H'} && \text{(Observation above)} \\
&\subseteq \left(\bigcup_{D \in r(C)} D[\llbracket r(f) \rrbracket] \right)^{H'} && \text{(Lemma 4.13)} \\
&= r(C[\llbracket f \rrbracket])^{H'} && \text{(Lemma 4.B.1)} \\
&\subseteq \left(r(L)^{H'} \right)^{H'} && \text{(Induction)} \\
&\subseteq r(L)^{H'} && \text{(Lemma 4.11)} \quad \square
\end{aligned}$$

To prove Lemma 4.40, we need the following technical properties.

Lemma 4.B.2. *Let $C \in \mathsf{PC}^{\mathsf{sp}}$ and $U \in \mathsf{Pom}$. If $C[U] \in \Sigma^*$ and $U \neq 1$, then $U \in \Sigma^*$, and there exist $w, x \in \Sigma^*$ such that $C = w \cdot \square \cdot x$.*

Proof. The proof proceeds by induction on the construction of C. In the base, where $C = \square$, we have $U = C[U] \in \Sigma^*$, and we can choose $w = x = 1$ to satisfy the claim. For the inductive step, there are three cases to consider.

- If $C = C' \cdot V$ for some $C' \in \mathsf{PC}^{\mathsf{sp}}$ and $V \in \mathsf{SP}$, then note that since $C[U]$ is totally ordered, so is $C'[U] \in \Sigma^*$. By induction, we then find that $U \in \Sigma^*$, and that $w', x' \in \Sigma^*$ such that $C' = w' \cdot \square \cdot x'$. We can then choose $w = w'$ and $x = x' \cdot V$ to satisfy the claim.
- If $C = V \cdot C'$ for some $C' \in \mathsf{PC}^{\mathsf{sp}}$ and $V \in \mathsf{SP}$, then an argument similar to the above applies.

- If $C = V \parallel C'$ for $C' \in \mathsf{PC}^{\mathsf{sp}}$ and $V \in \mathsf{SP}$, then since $C[U]$ is sequential and $C[U] = V \parallel C'[U]$, it follows that V must be empty (since $C'[U]$ cannot be empty). Hence, $C = C'$; by induction we find that $U \in \Sigma^*$, and we obtain $w, x \in \Sigma^*$ such that $C = C' = w \cdot \Box \cdot x$. $\Box$

Lemma 4.B.3. *Let $C \in \mathsf{PC}^{\mathsf{sp}}$ be a pomset context, let $V, W \in \mathsf{Pom}$, and let $x \in \Sigma^*$ be non-empty. If $C[x] = V \parallel W$, then there exists a $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that either $C = C' \parallel W$ and $C'[x] = V$, or $C = V \parallel C'$ and $C'[x] = W$.*

Proof. We proceed by induction on C. In the base, $C = \Box$, in which case $w = C[x] = V \parallel W$. Since x is a non-empty word, either $V = 1$ or $W = 1$ — otherwise, $C[x]$ is both sequential and parallel. In the former case, we choose $C' = C$ to find that $C'[U] = C[U] = U = W$, and $C = C \parallel 1 = C' \parallel V$; the latter case can be handled similarly. For the inductive step, there are three cases to consider.

- If $C = D \cdot X$ for some $D \in \mathsf{PC}^{\mathsf{sp}}$ and $X \in \mathsf{SP}$, then $D[x] \cdot X = V \parallel W$. Since $D[x]$ is non-empty and $C[x]$ cannot be both sequential and parallel, there are two subcases. If $V = 1$, then choose $C' = C$ such that $C'[x] = C[x] = W$ and $C = C \parallel 1 = C' \parallel V$. The case where $W = 1$ is similar. Otherwise, if $X = 1$, then $C[x] = D[x] = V \parallel W$. The claim follows by induction.
- If $C = X \cdot D$ for some $D \in \mathsf{PC}^{\mathsf{sp}}$ and $X \in \mathsf{SP}$, then we can find C' analogously to the above.
- If $C = D \parallel X$ with $D \in \mathsf{PC}^{\mathsf{sp}}$ and $X \in \mathsf{SP}$, then by Lemma 3.A.2 we obtain $Y_0, Y_1, Z_0, Z_1 \in \mathsf{SP}$ such that $D[x] = Y_0 \parallel Y_1$, $X = Z_0 \parallel Z_1$, $V = Y_0 \parallel Z_0$, and $W = Y_1 \parallel Z_1$. By induction, we find $D' \in \mathsf{PC}^{\mathsf{sp}}$ such that either $D = D' \parallel Y_1$ and $D'[x] = Y_0$, or $D = Y_0 \parallel D'$ and $D'[x] = Y_1$. In the former case, we can choose $C' = D' \parallel Z_0$ to find that $C'[x] = D'[x] \parallel Z_0 = Y_0 \parallel Z_0 = V$ and $C = D \parallel X = D' \parallel Y_1 \parallel Z_0 \parallel Z_1 = C' \parallel W$. The latter case is similar. $\Box$

Lemma 4.40. *Let H be grounded. If $L \subseteq \Sigma^*$, then $L^H = L^{\langle H \rangle}$.*

Furthermore, for $L, L' \subseteq \mathsf{SP}$, we have that $(L \parallel L')^H = L^H \parallel L'^H$.

Proof. For the first claim, we start with the inclusion from left to right. Here, we show that if $A \subseteq L^{\langle H \rangle}$, then $A \subseteq L^H$. To see this, we proceed by induction on the construction of $A \subseteq L^{\langle H \rangle}$. In the base, $A = L$, and hence $A \subseteq L^H$ immediately. For the inductive step, we obtain $e \leq f \in H$ and $x, y \in \Sigma^*$ with $A = x \cdot [\![e]\!] \cdot y$ and $w \cdot [\![f]\!] \cdot x \subseteq L^{\langle H \rangle}$. By induction, we know $x \cdot [\![f]\!] \cdot y \subseteq L^H$. We can then choose $C = x \cdot \Box \cdot y$ to find that, since $C[[\![f]\!]] \subseteq L^H$, we also have $x \cdot [\![e]\!] \cdot y = C[[\![e]\!]] \subseteq L^H$.

For the other inclusion, we proceed by induction on the construction of $A \subseteq L^H$, showing that $A \subseteq L^{\langle H \rangle}$ and $A \subseteq \Sigma^*$. In the base, we know that $A \subseteq L$, hence $A \subseteq L^{\langle H \rangle}$ and $A \subseteq \Sigma^*$ immediately. For the inductive step, we find $e \leq f \in H$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ such that $A = C[[\![e]\!]]$ and $C[[\![f]\!]] \subseteq L^H$.

Since H is grounded, we have $[\![f]\!] = \{w\}$ for some non-empty word w. Since $C[w] \in C[[\![f]\!]] \subseteq \Sigma^*$ by induction, it follows that $C = x \cdot \Box \cdot y$ for $x, y \in \Sigma^*$ by Lemma 4.B.2. Also by induction, we know that $x \cdot [\![f]\!] \cdot y = C[[\![f]\!]] \subseteq L^{\langle H \rangle}$; hence, $A = C[[\![e]\!]] = x \cdot [\![e]\!] \cdot y \subseteq L^H$. Finally, we note that since $e \in \mathcal{T}_\mathsf{R}$, we have that $[\![e]\!] \subseteq \Sigma^*$, and hence $A = C[[\![e]\!]] = x \cdot [\![e]\!] \cdot y \subseteq \Sigma^*$ as well.

For the second claim, the inclusion from right to left follows from Lemma 4.13. For the other inclusion, suppose $A \subseteq (L \parallel L')^H$; it suffices to show that we can find $B, B' \subseteq \mathsf{SP}$ such that $A \subseteq B \parallel B'$ and $B \subseteq L^H$ and $B' \subseteq L'^H$. We proceed by induction on the construction of $(L \parallel L')^H$. In the base, where $A \subseteq L \parallel L'$, we can choose $B = L$ and $B' = L'$ to satisfy the claim.

For the inductive step, $A \subseteq (L \parallel L')^H$ because there exist $C \in \mathsf{PC}^\mathsf{sp}$ and $e \leq f \in H$ with $A = C[[\![e]\!]]$ and $C[[\![f]\!]] \subseteq (L \parallel L')^H$. By induction, we find $B, B' \subseteq \mathsf{SP}$ with $C[[\![f]\!]] \subseteq B \parallel B'$ and $B \subseteq L^H$ and $B' \subseteq L'^H$. Since $e \leq f$ is grounded, $[\![f]\!] = \{w\}$ for some non-empty word w; hence $C[w] = X \parallel X'$ with $X \in B$ and $X' \in B'$. By Lemma 4.B.3, either $C = C' \parallel X'$ such that $C'[w] = X$, or $C = C' \parallel X$ such that $C'[w] = X'$. In the former case, $A = C[[\![e]\!]] \subseteq C'[[\![e]\!]] \parallel B'$. Since $C'[[\![e]\!]] \subseteq L^H$, the claim follows. The latter case can be treated similarly. $\square$

Lemma 4.46. *Let H_0, H_1 and H' be sets of hypotheses. If r is a reduction from H_0 to H', and r' is a reduction from H' to H_1, then $r' \circ r$ is a reduction from H_0 to H_1.*

Proof. Let r be the reduction from H to H', and let r' be the reduction from H' to H''. We claim that $r' \circ r$ is a reduction from H to H''. We check the conditions one-by-one.

To see that H implies H'', suppose that $e \leq f \in H''$. Since H' implies H'', we obtain $e \leqq^{H'} f$. Since H implies H', we find $e \leqq^H f$ by Lemma 4.23(i).

To see that $e \equiv^H r'(r(e))$, first note $e \equiv^H r(e)$. Also, $r(e) \equiv^{H'} r'(r(e))$, and since H implies H', we have $r(e) \equiv^H r'(r(e))$. The claim then follows by transitivity of $\equiv^H$.

Lastly, suppose that $e, f \in \mathcal{T}$ such that $[\![e]\!]^H = [\![f]\!]^H$. We then know that $[\![r(e)]\!]^{H'} = [\![r(f)]\!]^{H'}$, thus $[\![r'(r(e))]\!]^{H''} = [\![r'(r(e))]\!]^{H''}$, by definition of reduction. $\square$

Chapter 5

The Exchange Law

When writing program code, we typically want to abstract from the machine where the code is run. This is particularly true in the case of concurrency, where hardware constraints may mean that code specified to run in parallel is run sequentially. For instance, recall the tea dispenser discussed in Chapter 3. If the machine has only one outlet it may not be able to dispense milk and tea concurrently, because the steam coming off the hot tea may run back into the milk reservoir. In this case, it is still acceptable to first pour tea followed by milk, or vice versa (depending on the user's preference) — it still accomplishes the goal of serving tea with milk. Similarly, parallel threads in a program may be interleaved, for example when the number of threads exceeds the number of processors available, or when two threads need exclusive access to some resource.

For these reasons, the semantics of a programming language may include the possibility that behaviour is executed with more temporal dependencies than specified in the code. Hence, when reasoning algebraically about equivalence and containment of programs, we typically want to include the possibility that a (partial) sequentialisation is included in parallel composition of two threads. This is where the *exchange law* comes in [Gis88; HMS^{+}09]. We can encode the exchange law as an additional hypothesis on series-rational expressions, as follows.

Definition 5.1 (The exchange law as hypotheses)**.** The set of hypotheses exch is given by

$$\mathsf{exch} = \{(e \parallel f) \cdot (g \parallel h) \leq (e \cdot g) \parallel (f \cdot h) \ : \ e, f, g, h \in \mathcal{T}\}$$

Intuitively, the exchange law says that if two threads run in parallel, consisting of a first and second sequential part (as on the right hand side), then the first parts may be executed in parallel, followed by the second parts in parallel (as on the left hand side).

Example 5.2. Let $e, f \in \mathcal{T}$, and note that $e \cdot f \equiv (e \parallel 1) \cdot (1 \parallel f) \leqq^{\mathsf{exch}} (e \cdot 1) \parallel (1 \cdot f) \equiv e \parallel f$. In words, the behaviour of running e and f in parallel includes that of running e and f in sequence. We can apply this to the description of the tea dispenser in Example 3.49, where we find that

$$(\texttt{cup} \parallel \texttt{heat}) \cdot ((\texttt{dairy} + \texttt{soy} + 1) \cdot \texttt{tea}) \leqq^{\mathsf{exch}} (\texttt{cup} \parallel \texttt{heat}) \cdot ((\texttt{dairy} + \texttt{soy} + 1) \parallel \texttt{tea})$$

In this chapter, we study the exchange law as a set of hypotheses, and contribute two results.

- We prove that exch strongly reduces to $\emptyset$; this settles a conjecture by Hoare et al. [HSM+16], who proposed that $\equiv^{\mathsf{exch}}$ is sound and complete w.r.t. $[\![-]\!]^{\mathsf{exch}}$. This was also proved by Laurence and Struth [LS17]. In contrast with op. cit., we explicitly show how to compute this reduction, and we rely on syntactic constructions, rather than finite monoid theory.
- Second, we show that the exchange law can be factorised from certain hypotheses, enabling a modular approach to establish reductions of new sets of hypotheses that include exch.

Remark 5.3. The axioms that generate $\equiv^{\mathsf{exch}}$, i.e., those of bi-Kleene algebra in conjunction with the exchange law, are precisely the axioms of *concurrent Kleene algebra* [HMS+09].

Remark 5.4. The reader be reminded of *interchange law* from category theory [Mac98, pp. 43]. The exchange law, in contrast, is an inequation; had we used an equational exchange law, i.e.,

$$\mathsf{exch}' = \{(e \parallel f) \cdot (g \parallel h) = (e \cdot g) \parallel (f \cdot h) \ : \ e, f, g, h \in \mathcal{T}\}$$

then, using the Eckmann–Hilton argument [EH62], we could derive as follows:

$$e \cdot f \equiv (e \parallel 1) \cdot (1 \parallel f) \equiv^{\mathsf{exch}'} (e \cdot 1) \parallel (1 \cdot f) \equiv e \parallel f \equiv (1 \cdot e) \parallel (f \cdot 1) \equiv^{\mathsf{exch}'} (1 \parallel f) \cdot (e \parallel 1) \equiv f \cdot e$$

Thus, exch' would make sequential coincide with parallel composition, which makes little sense.

Since exch encodes the possibility of partial interleaving, and subsumption relates a pomset to a "more sequential" version, this begs the question: can subsumption be related to the exchange law? In fact, the exchange law also holds in the level of pomsets and subsumption.

Lemma 5.5 [Gis88]. *Let $\sqsubseteq^{\mathsf{sp}}$ be $\sqsubseteq$ restricted to SP. Then $\sqsubseteq^{\mathsf{sp}}$ is the smallest precongruence satisfying the exchange law, i.e., such that for all $U, V, W, X \in \mathsf{SP}$, it holds that*

$$(U \parallel V) \cdot (W \parallel X) \sqsubseteq^{\mathsf{sp}} (U \cdot W) \parallel (V \cdot X)$$

Lemma 5.5 allows us to show that closure w.r.t. the set of hypotheses exch is the same as downward closure w.r.t. subsumption restricted to series-parallel pomsets. This shows that the exch-closed semantics of sr-expressions coincides precisely with the semantics proposed in [HMS+09].

Corollary 5.6. *Let $L \subseteq \mathsf{SP}$. Now $U \in L^{\mathsf{exch}}$ if and only if there exists a $V \in L$ such that $U \sqsubseteq V$.*

In the sequel, we may write $L{\downarrow}$ for the smallest series-parallel language containing L such that if $U \sqsubseteq V \in L{\downarrow}$, then $U \in L{\downarrow}$. By the above, this set is the same as L^{exch}.

5.1 Reduction

We start by showing that exch strongly reduces to the empty set of hypotheses. This means that we should show that, for every sr-expression e, we can compute an sr-expression that is equivalent under exch, but whose semantics already includes the pomsets added by exch-closure. By Corollary 5.6, we can then simplify the requirements on the expression that we need to compute to the following.

Definition 5.7 (Closure)**.** Let $e \in \mathcal{T}$; $e{\downarrow} \in \mathcal{T}$ is a *closure* of e if $e{\downarrow} \leqq^{\mathsf{exch}} e \leqq e{\downarrow}$ and $[\![e{\downarrow}]\!] = [\![e]\!]{\downarrow}$.

Clearly, if for every $e \in \mathcal{T}$ we can compute a closure $e{\downarrow}$, we have a reduction from exch to $\emptyset$. Note that a closure of $e \in \mathcal{T}$, if it exists, is unique up to $\equiv^{\mathsf{exch}}$-equivalence, by definition. Going forward, we will therefore speak of *the* closure of a series-rational expression.

Example 5.8. The closure of $e = \mathtt{a} \parallel \mathtt{b}$ is $e{\downarrow} = \mathtt{a} \parallel \mathtt{b} + \mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}$, where $\mathtt{a}$ and $\mathtt{b}$ execute in parallel, or either precedes the other — matching the optional parallelism. Since $\mathtt{a} \cdot \mathtt{b}$, $\mathtt{b} \cdot \mathtt{a} \leqq^{\mathsf{exch}} \mathtt{a} \parallel \mathtt{b}$, we have that $e{\downarrow} \leqq^{\mathsf{exch}} e$; furthermore, $e \leqq e{\downarrow}$ by construction. If $U \in [\![e]\!]$, then $U = \mathtt{a} \parallel \mathtt{b}$; the pomsets subsumed by U are U itself, $\mathtt{a} \cdot \mathtt{b}$ and $\mathtt{b} \cdot \mathtt{a}$, which are precisely the pomsets in $[\![e{\downarrow}]\!]$.

Example 5.9. A closure of $f = \mathtt{a}^* \parallel \mathtt{b}^*$ would be $f{\downarrow} = (\mathtt{a}^* \parallel \mathtt{b}^*)^*$. To prove this, first note that $f \leqq 1 + f \cdot f^* \equiv f^* = f{\downarrow}$. To show that $f{\downarrow} \leqq^{\mathsf{exch}} f$, we start by deriving that

$$(\mathtt{a}^* \parallel \mathtt{b}^*) \cdot (\mathtt{a}^* \parallel \mathtt{b}^*) + 1 \leqq^{\mathsf{exch}} \mathtt{a}^* \cdot \mathtt{a}^* \parallel \mathtt{b}^* \cdot \mathtt{b}^* + 1 \equiv \mathtt{a}^* \parallel \mathtt{b}^* + 1 \equiv \mathtt{a}^* \parallel \mathtt{b}^*$$

where in the second step we use the fact that, for all $g \in \mathcal{T}$, it holds that $g^* \cdot g^* \equiv g^*$. From this, we can conclude that $f{\downarrow} \equiv (\mathtt{a}^* \parallel \mathtt{b}^*)^* \cdot 1 \leqq \mathtt{a}^* \parallel \mathtt{b}^* = f$. We can also show $[\![f]\!]{\downarrow} = [\![f{\downarrow}]\!]$, if we argue that when $U \sqsubseteq^{\mathsf{sp}} V \in [\![\mathtt{a}^* \parallel \mathtt{b}^*]\!]$, also $U \in [\![(\mathtt{a}^* \parallel \mathtt{b}^*)^*]\!]$; the proof is left as an exercise for the reader.

The remainder of this section is dedicated to describing exactly how such a closure can be computed. To this end, we take an inductive approach, in that we compute the closure of an sr-expression e using (possibly several) closures already computed. More precisely, we assume that we can compute closures of sr-expressions e' where $\parallel$ is nested less often than e, and use those to compute the closure of e proper. This is formalised as follows.

$$\begin{tikzcd} \boxed{\langle e, f\rangle} \arrow[r, "e_0 \odot f_0"] & \boxed{\langle e', f'\rangle} \arrow[r, "e'_0 \odot f'_0"] & \boxed{\langle e'', f''\rangle} \arrow[r, "e''_0 \odot f''_0"] & \cdots \end{tikzcd}$$

Figure 5.1: The series-rational system to compute closure of $e \parallel f$.

Definition 5.10 (∥-depth)**.** We denote the ∥-*depth* of $e \in \mathcal{T}$ by $\mathsf{d}_\parallel(e)$, i.e.,

$$\begin{array}{llll} \mathsf{d}_\parallel(0) = 0 & \mathsf{d}_\parallel(\mathtt{a}) = 0 & \mathsf{d}_\parallel(e \cdot f) = \max(\mathsf{d}_\parallel(e), \mathsf{d}_\parallel(f)) & \mathsf{d}_\parallel(e^*) = \mathsf{d}_\parallel(e) \\ \mathsf{d}_\parallel(1) = 0 & \mathsf{d}_\parallel(e + f) = \max(\mathsf{d}_\parallel(e), \mathsf{d}_\parallel(f)) & \mathsf{d}_\parallel(e \parallel f) = \max(\mathsf{d}_\parallel(e), \mathsf{d}_\parallel(f)) + 1 & \end{array}$$

Our *main induction hypothesis* in computing a closure of e will be the following

If $f \in \mathcal{T}$ and $\mathsf{d}_\parallel(f) < \mathsf{d}_\parallel(e)$, then we can compute $f{\downarrow}$, the closure of f.

We want to show that, under the main induction hypothesis, we can compute a closure of e. To this end, we observe the following property of subsumption closure and pomset language composition.

Lemma 5.11 (c.f. [Gis88, Theorems 5.2 and 5.4])**.** *Let* $L, K \subseteq \mathsf{SP}$ *and* $\mathtt{a} \in \Sigma$. *The following hold:*

$$\{1\}{\downarrow} = \{1\} \qquad \{\mathtt{a}\}{\downarrow} = \{\mathtt{a}\} \qquad (L \cup K){\downarrow} = L{\downarrow} \cup K{\downarrow} \qquad (L \cdot K){\downarrow} = L{\downarrow} \cdot K{\downarrow} \qquad L^*{\downarrow} = (L{\downarrow})^*$$

The above can be interpreted to mean that closure can be computed using a divide-and-conquer approach for all operators except parallel composition. More precisely, if we want to compute the closure of an sr-expression like $e \cdot f$, then we can simply compute the closures of e and f individually, and output $e{\downarrow} \cdot f{\downarrow}$. After all, $e \cdot f \equiv^{\mathsf{exch}} e{\downarrow} \cdot f{\downarrow}$ by congruence, and using Lemma 5.11 we can derive

$$[\![e \cdot f]\!]{\downarrow} = ([\![e]\!] \cdot [\![f]\!]){\downarrow} = [\![e]\!]{\downarrow} \cdot [\![f]\!]{\downarrow} = [\![e{\downarrow}]\!] \cdot [\![f{\downarrow}]\!] = [\![e{\downarrow} \cdot f{\downarrow}]\!]$$

Similar observations hold for expressions of the form $e + f$ and e^*. The above lemma also implies that expressions of the form 0, 1, or $\mathtt{a}$ for some $\mathtt{a} \in \Sigma$ are their own closure. The recursive computation of closures can continue, until we hit an expression of the form $e \parallel f$; it remains to find a closure of this expression. This is where the heavy lifting of closure computation will be done.

Before we go on, recall the intuition behind the exchange law: if we can run some "initial" parts of e and f in parallel, and continue by running the "remaining" parts of the operands in parallel, that is a valid behaviour of $e \parallel f$. We will exploit this by creating an operational representation in the form of a series-rational system. In this representation, the state $\langle e, f\rangle$ with $e \equiv e_0 \cdot e'$ and $f \equiv f_0 \cdot f'$, can perform an interleaving of e_0 in parallel with f_0 to transition into the state $\langle e', f'\rangle$ (c.f. Figure 5.1). The idea is that the solution to $\langle e, f\rangle$ in the sr-system will be the closure of $e \parallel f$.

This sketch leaves us with two things to formalize:

- The sr-system discussed above needs to be constructed explicitly. In particular, we need to show that it is finite, as well as how to find the different ways of splitting e and f into initial and remaining parts. This will be done in Section 5.1.2.
- Some interleaving that is internal to the parallel composition of e_0 and f_0 may still take place. We need to use closures of simpler expressions to calculate an expression that encodes this interleaving, to label the transition into the next state. This is done in Section 5.1.1.

5.1.1 Preclosure

We start by working out what the labels for the transitions in our sr-system to compute closure should be. Recall that a transition in this system is the parallel composition of some initial parts e_0 and f_0 of both operands, and that we should work out some interleaving between them. To this end, we will assume that the main induction hypothesis holds for $e_0 \parallel f_0$, i.e., that for all $g \in \mathcal{T}$ such that $\mathsf{d}_\parallel(g) < \mathsf{d}_\parallel(e_0 \parallel f_0)$ we can compute a closure $g{\downarrow}$. Thus, we cannot just take the closure of $e_0 \parallel f_0$ as the label of this transition, because it cannot be strictly simpler than itself. On the other hand, we are interested only in the pomsets in $[\![e_0 \parallel f_0]\!]{\downarrow}$ that represent a single "step" — after all, a pomset that represents multiple sequential steps should be represented by multiple transitions in sequence. Hence, we do not need a full closure of $e_0 \parallel f_0$. Instead, we make do with the following.

Definition 5.12 (Preclosure)**.** Let $e \in \mathcal{T}$. A *preclosure* of e is an sr-expression $\tilde{e} \in \mathcal{T}$ such that $\tilde{e} \leqq^{\mathsf{exch}} e$ and $e \leqq \tilde{e}$, and if $U \in [\![e]\!]{\downarrow}$ is a sequential prime (c.f. Convention 3.18), then $U \in [\![\tilde{e}]\!]$.

Example 5.13. We start with a non-example. Let $e_0 = \mathtt{a} \parallel \mathtt{b}$ and $f_0 = \mathtt{c}$. Recall from Example 5.8 that $e_0{\downarrow} = \mathtt{a} \parallel \mathtt{b} + \mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}$ is a closure of e_0, and observe that $f_0{\downarrow} = \mathtt{c}$ is a closure of f_0. Now, to compute a preclosure of $e_0 \parallel f_0$, we cannot simply take the parallel composition of these closures, because $\mathtt{a} \cdot \mathtt{c} \parallel \mathtt{b} \in [\![e_0 \parallel f_0]\!]{\downarrow}$ is a sequential prime, while $\mathtt{a} \cdot \mathtt{c} \parallel \mathtt{b} \notin [\![e_0{\downarrow} \parallel f_0{\downarrow}]\!]$.

Example 5.14. Let $e_0 = \mathtt{a} \parallel \mathtt{b}$ and $f_0 = \mathtt{c}$, as above. A valid preclosure of $e_0 \parallel f_0$ could be

$$g = \mathtt{a} \parallel \mathtt{b} \parallel \mathtt{c} + (\mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}) \parallel \mathtt{c} + (\mathtt{b} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{b}) \parallel \mathtt{a} + (\mathtt{a} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{a}) \parallel \mathtt{b}$$

To verify this, first note that $e_0 \parallel f_0 \leqq g$ by construction. It remains to show that $g \leqq^{\mathsf{exch}} e_0 \parallel f_0$. This is fairly straightforward: since $\mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a} \leqq^{\mathsf{exch}} \mathtt{a} \parallel \mathtt{b}$, we have $(\mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}) \parallel \mathtt{c} \leqq^{\mathsf{exch}} e_0 \parallel f_0$; a similar remark holds for the other expressions. Since all the terms of g are below $e_0 \parallel f_0$, so is g itself. Furthermore, there are seven non-sequential and non-empty pomsets in $[\![e_0 \parallel f_0]\!]{\downarrow}$; they are

$$\mathtt{a} \parallel \mathtt{b} \parallel \mathtt{c} \qquad \mathtt{a} \cdot \mathtt{b} \parallel \mathtt{c} \qquad \mathtt{b} \cdot \mathtt{a} \parallel \mathtt{c} \qquad \mathtt{b} \cdot \mathtt{c} \parallel \mathtt{a} \qquad \mathtt{c} \cdot \mathtt{b} \parallel \mathtt{a} \qquad \mathtt{a} \cdot \mathtt{c} \parallel \mathtt{b} \qquad \mathtt{c} \cdot \mathtt{a} \parallel \mathtt{b}$$

Each of these pomsets is found in $[\![g]\!]$. It should be noted that g is *not* a closure of e; to see this, observe for instance that $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \in [\![e_0 \parallel f_0]\!]{\downarrow}$, while $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \not\in [\![g]\!]$, and hence $[\![e_0 \parallel f_0]\!]{\downarrow} \neq [\![g]\!]$.

Thus, our focus is now on computing a preclosure of sr-expressions of the form $e \parallel f$. The examples above already give us a hint: we cannot simply take the closures of e and f and put them in parallel; if we do that, we miss out on possible sequential primes that can be obtained by interleaving a parallel part of e with f, and vice versa, such as the pomset $\mathtt{a} \cdot \mathtt{c} \parallel \mathtt{b}$ discussed there.

Instead, we need to compute the closures of each (non-trivial) parallel part of $e \parallel f$, and accumulate them. This is how one can arrive at the preclosure in the second example, which includes for instance a closure of $\mathtt{a} \parallel \mathtt{c}$ in parallel with $\mathtt{b}$, even though $\mathtt{a}$ is a parallel part of e_0, while $\mathtt{c}$ is the whole of f_0. To obtain the parallel parts of an sr-expression, we propose the following.

Definition 5.15 (Parallel splitting)**.** We define Δ as the smallest subset of $\mathcal{T} \times \binom{\mathcal{T}}{2}$ that satisfies the following rules, where $e, f \in \mathcal{T}$, and we abbreviate $\langle e, \{\!|\ell, r|\!\}\rangle \in \Delta$ by writing $\ell \mathrel{\Delta_e} r$.

$$\frac{}{e \mathrel{\Delta_{e \parallel f}} f} \qquad \frac{\ell \mathrel{\Delta_e} r \qquad \ell' \mathrel{\Delta_f} r'}{\ell \parallel \ell' \mathrel{\Delta_{e \parallel f}} r \parallel r'} \qquad \frac{\ell \mathrel{\Delta_e} r}{\ell \mathrel{\Delta_{e \parallel f}} r \parallel f} \qquad \frac{\ell \mathrel{\Delta_f} r}{e \parallel \ell \mathrel{\Delta_{e \parallel f}} r}$$

$$\frac{\ell \mathrel{\Delta_e} r}{\ell \mathrel{\Delta_{e+f}} r} \qquad \frac{\ell \mathrel{\Delta_f} r}{\ell \mathrel{\Delta_{e+f}} r} \qquad \frac{\ell \mathrel{\Delta_e} r}{\ell \mathrel{\Delta_{e^*}} r} \qquad \frac{\ell \mathrel{\Delta_e} r \qquad f \in \mathcal{F}}{\ell \mathrel{\Delta_{e \cdot f}} r} \qquad \frac{\ell \mathrel{\Delta_f} r \qquad e \in \mathcal{F}}{\ell \mathrel{\Delta_{e \cdot f}} r}$$

When $\ell \mathrel{\Delta_e} r$, we say that ℓ and r form a *parallel splitting* of e.

The final two rules are meant to accommodate the fact that the empty pomset is neutral for sequential composition, and hence if we can split e in parallel into ℓ and r, then the same holds for $e \cdot f$, provided that the semantics of f contains the empty pomset (i.e., $f \in \mathcal{F}$; c.f. Definition 3.52).

Remark 5.16. It may not be immediately obvious from the notation, but for every $e \in \mathcal{T}$, the "relation" Δ_e is symmetric; after all, if $\ell \mathrel{\Delta_e} r$, then $\langle e, \{\!|r, \ell|\!\}\rangle = \langle e, \{\!|\ell, r|\!\}\rangle \in \Delta$, and hence $r \mathrel{\Delta_e} \ell$.

Example 5.17. Let $e_0 = \mathtt{a} \parallel \mathtt{b}$ and $f_0 = \mathtt{c}$, as in Example 5.14. Since $\mathtt{a} \mathrel{\Delta_{e_0}} \mathtt{b}$ by the first rule above, it follows that $\mathtt{b} \mathrel{\Delta_{e_0}} \mathtt{a}$, and hence we find that $\mathtt{b} \mathrel{\Delta_{e_0 \parallel f_0}} \mathtt{a} \parallel \mathtt{c}$, by the third rule above.

Using the above example, we see that the pomset $\mathtt{a} \parallel \mathtt{b} \parallel \mathtt{c}$, which appears in $[\![e_0 \parallel f_0]\!]$, can be split into the parallel parts $\mathtt{b}$ and $\mathtt{a} \parallel \mathtt{c}$, which can be found in the semantics of $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\Delta_e} r$. To validate the intuition that this can be done in general, i.e., that all possible $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\Delta_e} r$ cover the ways of splitting the pomsets in e, we record the following lemma.

Lemma 5.18. *Let $e \in \mathcal{T}$. If $V, W \in \mathsf{SP}$ are non-empty and $V \parallel W \in [\![e]\!]$, then there exist $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\Delta_e} r$ such that $V \in [\![\ell]\!]$ and $W \in [\![r]\!]$.*

We are now almost ready to define the preclosure of $e \parallel f$. Before we can, however, we need to note some technical properties of the parallel splitting relation, recorded in the following lemma.

Lemma 5.19. *Let $e \in \mathcal{T}$. The following hold:*

(i) There are finitely many $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\Delta_e} r$.

(ii) If $\ell \mathrel{\Delta_e} r$, then $\ell \parallel r \leqq e$.

(iii) If $\ell \mathrel{\Delta_e} r$, then $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r) < \mathsf{d}_{\parallel}(e)$.

The preclosure of a parallel composition can then be constructed as follows.

Definition 5.20 (Syntactic construction of preclosure)**.** Let $e, f \in \mathcal{T}$, and suppose that the main induction hypothesis applies to $e \parallel f$. The sr-expression $e \odot f$ is defined to be

$$e \odot f = \sum_{\ell \Delta_{e \parallel f} r} \ell{\downarrow} \parallel r{\downarrow}$$

Remark 5.21. Note that $e \odot f$ is well-defined and computable: if $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\Delta_{e\parallel f}} r$, then $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r) < \mathsf{d}_{\parallel}(e \parallel f)$ by Lemma 5.19(iii), and hence the sr-expressions $\ell{\downarrow}$ and $r{\downarrow}$ exist and are computable. Also, since there are finitely many such ℓ, r by Lemma 5.19(i), the sum is finite.

Example 5.22. Let $e_0 = \mathtt{a} \parallel \mathtt{b}$ and $f_0 = \mathtt{c}$. We compute $e_0 \odot f_0$ and verify that we obtain a preclosure of $e_0 \parallel f_0$. Working through the definition, we see that if $\ell \mathrel{\Delta_{e_0 \parallel f_0}} r$, then $\{\!|\ell, r|\!\}$ must be one of $\{\!|\mathtt{a} \parallel \mathtt{b}, \mathtt{c}|\!\}$ $\{\!|\mathtt{a}, \mathtt{b} \parallel \mathtt{c}|\!\}$, or $\{\!|\mathtt{b}, \mathtt{a} \parallel \mathtt{c}|\!\}$. Recall that $\mathtt{a} \parallel \mathtt{b} + \mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}$ is a closure of $\mathtt{a} \parallel \mathtt{b}$, and note that the closures of $\mathtt{b} \parallel \mathtt{c}$ and $\mathtt{a} \parallel \mathtt{c}$ are computed analogously. Now, we find that

$$\begin{aligned}
e_0 \odot f_0 &= (\mathtt{a} \parallel \mathtt{b}){\downarrow} \parallel \mathtt{c} + \mathtt{a} \parallel (\mathtt{b} \parallel \mathtt{c}){\downarrow} + \mathtt{b} \parallel (\mathtt{a} \parallel \mathtt{c}){\downarrow} \\
&= (\mathtt{a} \parallel \mathtt{b} + \mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}) \parallel \mathtt{c} + \mathtt{a} \parallel (\mathtt{b} \parallel \mathtt{c} + \mathtt{b} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{b}) + \mathtt{b} \parallel (\mathtt{a} \parallel \mathtt{c} + \mathtt{a} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{a}) \\
&\equiv \mathtt{a} \parallel \mathtt{b} \parallel \mathtt{c} + (\mathtt{a} \cdot \mathtt{b} + \mathtt{b} \cdot \mathtt{a}) \parallel \mathtt{c} + (\mathtt{b} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{b}) \parallel \mathtt{a} + (\mathtt{a} \cdot \mathtt{c} + \mathtt{c} \cdot \mathtt{a}) \parallel \mathtt{b}
\end{aligned}$$

which was shown to be a preclosure of $e_0 \parallel f_0$ in Example 5.14.

We now have everything in place to formally verify that $e_0 \odot f_0$ is indeed a preclosure of $e_0 \parallel f_0$.

Lemma 5.23. *Let $e, f \in \mathcal{T}$, and suppose that the main induction hypothesis applies to $e \parallel f$. Then $e \odot f$ is the preclosure of $e \parallel f$.*

Proof. Note that $e{\downarrow} \parallel f{\downarrow}$ appears in the sum that builds $e \odot f$; hence, $e \parallel f \leqq e{\downarrow} \parallel f{\downarrow} \leqq e \odot f$. For the other inclusion, note that if $\ell{\downarrow} \parallel r{\downarrow}$ is a term in the sum that builds $e \odot f$, then $\ell{\downarrow} \parallel r{\downarrow} \leqq^{\mathsf{exch}} \ell \parallel r \leqq e \parallel f$, where the last step follows by Lemma 5.19(ii). Hence, $e \odot f \leqq^{\mathsf{exch}} e \parallel f$ as well.

Finally, suppose that $U \in [\![e \parallel f]\!]{\downarrow}$ is a sequential prime. Then there exists a $U' \in [\![e \parallel f]\!]$ such that $U \sqsubseteq U'$. On the one hand, if U is primitive, then $U = U'$, by Lemma 3.26, and hence

$$U \in [\![e \parallel f]\!] \subseteq [\![e{\downarrow} \parallel f{\downarrow}]\!] \subseteq [\![e \odot f]\!]$$

Otherwise, if U is parallel, then $U = V \parallel W$ such that V and W are non-empty. By Lemma 3.27, $U' = V' \parallel W'$ such that $V \sqsubseteq V'$ and $W \sqsubseteq W'$, with both V' and W' non-empty. By Lemma 5.18, we then find $\ell, r \in \mathcal{T}$ such that $V' \in [\![\ell]\!]$ and $W' \in [\![r]\!]$, with $\ell \; \Delta_{e\parallel f} \; r$. Hence, $V \in [\![\ell]\!]{\downarrow} = [\![\ell{\downarrow}]\!]$ and similarly $W \in [\![r{\downarrow}]\!]$. This allows us to conclude that $U = V \parallel W \in [\![\ell{\downarrow} \parallel r{\downarrow}]\!] \subseteq [\![e \odot f]\!]$. □

5.1.2 Closure

We now focus our attention on constructing the sr-system that represents the operational perspective on the closure of an sr-expression like $e \parallel f$. Here, we assume that the main induction hypothesis applies to $e \parallel f$, i.e., if $g \in \mathcal{T}$ such that $\mathsf{d}_{\parallel}(g) < \mathsf{d}_{\parallel}(e \parallel f)$, then we can compute the closure of g.

Recall that the sketch of the proposed series-rational system relied on taking some "initial" parts e_0 and f_0 of e and f respectively, computing the preclosure of this part, and jumping to the parallel composition of the "remaining" parts e' and f' of e and f. We therefore need a method to compute the ways in which we can split e and f into initial and remaining parts. We formalise this analogously to the parallel splitting of sr-expressions that we needed to define preclosure.

Definition 5.24 (Sequential splitting)**.** We define ∇ as the smallest subset of $\mathcal{T}^3$ that satisfies the following rules, where $\mathtt{a} \in \Sigma$ and $e, f \in \mathcal{T}$, and we abbreviate $\langle e, \ell, r\rangle \in \nabla$ by writing $\ell \; \nabla_e \; r$.

$$\frac{}{1 \; \nabla_1 \; 1} \qquad \frac{}{\mathtt{a} \; \nabla_{\mathtt{a}} \; 1} \qquad \frac{}{1 \; \nabla_{\mathtt{a}} \; \mathtt{a}} \qquad \frac{}{1 \; \nabla_{e^*} \; 1} \qquad \frac{\ell \; \nabla_e \; r}{\ell \; \nabla_{e+f} \; r} \qquad \frac{\ell \; \nabla_f \; r}{\ell \; \nabla_{e+f} \; r}$$

$$\frac{\ell \; \nabla_e \; r}{\ell \; \nabla_{e \cdot f} \; r \cdot f} \qquad \frac{\ell \; \nabla_f \; r}{e \cdot \ell \; \nabla_{e \cdot f} \; r} \qquad \frac{\ell \; \nabla_e \; r \qquad \ell' \; \nabla_f \; r'}{\ell \parallel \ell' \; \nabla_{e \parallel f} \; r \parallel r'} \qquad \frac{\ell \; \nabla_e \; r}{e^* \cdot \ell \; \nabla_{e^*} \; r \cdot e^*}$$

When $\ell \; \nabla_e \; r$, we say that ℓ and r are a *sequential splitting* of e.

Example 5.25. Let $e = \mathtt{a}^*$ and $f = \mathtt{b} \parallel \mathtt{c}$. Since we have $\mathtt{a} \; \nabla_{\mathtt{a}} \; 1$ by the third rule, it follows that $\mathtt{a}^* \cdot \mathtt{a} \; \nabla_e \; 1 \cdot \mathtt{a}^*$, by the last rule. Furthermore, since $1 \; \nabla_{\mathtt{b}} \; \mathtt{b}$ and $\mathtt{c} \; \nabla_{\mathtt{c}} \; 1$ by the second and third rule, we have that $1 \parallel \mathtt{c} \; \nabla_f \; \mathtt{b} \parallel \mathtt{1}$ by the second-to-last rule. If we apply the second-to-last rule to those splittings of e and f, we may then conclude that $\mathtt{a}^* \cdot \mathtt{a} \parallel \mathtt{1} \parallel \mathtt{c} \; \nabla_{e\parallel f} \; 1 \cdot \mathtt{a}^* \parallel \mathtt{b} \parallel 1$.

The intuition behind the sequential splitting relation is that for every way of dividing a pomset $U \in [\![e]\!]{\downarrow}$ into sequential parts, we can find $\ell, r \in \mathcal{T}$ such that $\ell \; \nabla_e \; r$, and $[\![r]\!]{\downarrow}$ and $[\![\ell]\!]{\downarrow}$ contain the first and second sequential parts of U. In other words, every way of splitting a pomset in the semantic closure of e can be traced to sequential parts of e. We formalise this in the next lemma.

Lemma 5.26. *Let $e \in \mathcal{T}$, and let V and W be pomsets such that $V \cdot W \in \llbracket e \rrbracket{\downarrow}$. Then there exist $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\nabla_e} r$ such that $V \in \llbracket \ell \rrbracket{\downarrow}$ and $W \in \llbracket r \rrbracket{\downarrow}$.*

Remark 5.27. Although sequential splitting is largely analogous to parallel splitting, Lemma 5.26 is not entirely analogous to Lemma 5.18: whereas the latter stipulates that V and W be non-empty, the former does not. This is a conscious choice in the definition of ∇: either of the "initial" parts of e and f may be empty, and hence our way of splitting these sr-expressions needs to account for this.

There is nothing conceptually preventing the "initial" part of e to be e itself, while the remaining part is 1. This is not directly true for sequential splitting; for instance, $\mathtt{a}^* \mathrel{\nabla_{\mathtt{a}^*}} 1$ does not hold. However, we can reconstruct e from the left-hand sides of finitely many splittings.

Lemma 5.28. *Let $e \in \mathcal{T}$. There exist $\ell_1, \dots, \ell_n \in \mathcal{T}$ and $r_1, \dots, r_n \in \mathcal{F}$ such that for $1 \leq i \leq n$ it holds that $\ell_i \mathrel{\nabla_e} r_i$, and furthermore $e \equiv \ell_1 + \cdots + \ell_n$.*

We also remark on some other technical properties of sequential splitting: that an sr-expression can be split in only finitely many pairs, that each of those pairs composed sequentially is contained in the original sr-expression (modulo exch) and that the splittings of e are at most as complex as e.

Lemma 5.29. *Let $e \in \mathcal{T}$. The following hold.*

(i) There are finitely many $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\nabla_e} r$.

(ii) If $\ell \mathrel{\nabla_e} r$, then $\ell \cdot r \leqq^{\mathsf{exch}} e$.

(iii) If $\ell \mathrel{\nabla_e} r$, then $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r) \leq \mathsf{d}_{\parallel}(e)$.

Next, we note that a state representing the closure of $e \parallel f$ should be able to visit only finitely many states. If this is not the case, then the sr-system may not have a solution. The right-hand sides that can be obtained by splitting an sr-expression e repeatedly are formalised as follows.

Definition 5.30. Let $e \in \mathcal{T}$. The set of *(right-hand) remainders* of e, written $R(e)$, is the smallest set containing e such that if $f \in R(e)$ and $\ell \mathrel{\nabla_f} r$, then $r \in R(e)$.

Example 5.31. Let $e = \mathtt{a}^*$. In Example 5.25, we found that $\mathtt{a}^* \cdot \mathtt{a} \mathrel{\nabla_e} 1 \cdot \mathtt{a}^*$; hence, $1 \cdot \mathtt{a}^* \in R(e)$. Furthermore, since $\mathtt{a}^* \cdot 1 \mathrel{\nabla_{\mathtt{a}^*}} \mathtt{a} \cdot \mathtt{a}^*$, we find that $1 \cdot \mathtt{a}^* \cdot 1 \mathrel{\nabla_{1 \cdot \mathtt{a}^*}} \mathtt{a} \cdot \mathtt{a}^*$ by the third-to-last rule of sequential splitting. Hence, by the second inference rule above, $\mathtt{a} \cdot \mathtt{a}^* \in R(e)$. Lastly, $1 \mathrel{\nabla_e} 1$ by the fourth rule defining sequential splitting. Splitting these sr-expressions further yields only remainders that we have already seen; hence, these comprise all of $R(e)$, that is, $R(e) = \{\mathtt{a}^*, 1 \cdot \mathtt{a}^*, \mathtt{a} \cdot \mathtt{a}^*, 1\}$.

In the example above, we remarked that $\mathtt{a}^*$ has finitely many right-hand remainders. This turns out to be true in general; moreover, each of those right-hand remainders is at most as complex as e.

Lemma 5.32. *Let $e \in \mathcal{T}$. $R(e)$ is finite. Furthermore, if $e' \in R(e)$, then $\mathsf{d}_{\parallel}(e') \leq \mathsf{d}_{\parallel}(e)$.*

We now have all of the ingredients to define the series-rational system that we were aiming for. In this system, each state is an expression of the form $g \parallel h$, with $g \in R(e)$ and $h \in R(f)$; it can either terminate and read the empty pomset provided g and h are accepting, or transition to a state $g' \parallel h'$ while performing an action from $g_0 \odot h_0$, provided that g splits into g_0 followed by g', and h splits into h_0 followed by h'. Since this system is series-rational, Theorem 3.60 allows us to compute a solution that is valid w.r.t. both $\equiv$ and $\equiv^{\mathsf{exch}}$. Formally, we define the system as follows.

Definition 5.33. Let $e, f \in \mathcal{T}$, and suppose that the main induction hypothesis applies to $e \parallel f$. We define a system $\mathcal{S} = \langle M, b \rangle$ on $Q = \{g \parallel h \;:\; g \in R(e), h \in R(f)\}$, with components given by

$$M(g \parallel h, g' \parallel h') = \sum_{\substack{g_0 \Delta_g g' \\ h_0 \Delta_h h'}} g_0 \odot h_0 \qquad\qquad b(g \parallel h) = [g \parallel h \in \mathcal{F}]$$

Let s be the least solution to $\mathcal{S}$ obtained through Theorem 3.60; we write $e \otimes f$ for $s(e \parallel f)$.

Remark 5.34. The system above is well-defined. We know that $R(e)$ and $R(f)$ are finite, by Lemma 5.32. Therefore, the system has finitely many variables. Moreover, for each $g' \in R(e)$ and $h' \in R(f)$, there are finitely many $g_0, h_0 \in \mathcal{T}$ such that $g_0 \nabla_g g'$ and $h_0 \nabla_h h'$, by Lemma 5.29(i) — hence, the sum in each component of M is finite. Lastly, we know that for each of those g_0 and h_0, we have $\mathsf{d}_{\parallel}(g_0) \leq \mathsf{d}_{\parallel}(g) \leq \mathsf{d}_{\parallel}(e)$ and $\mathsf{d}_{\parallel}(h_0) \leq \mathsf{d}_{\parallel}(h) \leq \mathsf{d}_{\parallel}(f)$, by Lemma 5.29(iii) and Lemma 5.32. Hence $\mathsf{d}_{\parallel}(g_0 \parallel h_0) \leq \mathsf{d}_{\parallel}(e \parallel f)$, meaning that the main induction hypothesis applies to $g_0 \parallel h_0$.

Example 5.35. Let $e = \mathtt{a}^*$ and $f = \mathtt{b}$. In Example 5.31, we saw that $R(e) = \{\mathtt{a}^*, 1 \cdot \mathtt{a}^*, \mathtt{a} \cdot \mathtt{a}^*, 1\}$; it is not hard to compute that $R(f) = \{\mathtt{b}, 1\}$. Suppose s is a solution to the series-rational system for $e \parallel f$. Unrolling the definitions, we then obtain the following constraint on $s(e \parallel f)$:

$$\begin{aligned} 0 \;&+\; (\mathtt{a}^* \cdot \mathtt{a} \odot 1) \cdot s(1 \cdot \mathtt{a}^* \parallel \mathtt{b}) \;+\; (\mathtt{a}^* \cdot 1 \odot 1) \cdot s(\mathtt{a} \cdot \mathtt{a}^* \parallel \mathtt{b}) \;+\; (1 \odot 1) \cdot s(1 \parallel \mathtt{b}) \\ &+\; (\mathtt{a}^* \cdot \mathtt{a} \odot \mathtt{b}) \cdot s(1 \cdot \mathtt{a}^* \parallel 1) \;+\; (\mathtt{a}^* \cdot 1 \odot \mathtt{b}) \cdot s(\mathtt{a} \cdot \mathtt{a}^* \parallel 1) \;+\; (1 \odot \mathtt{b}) \cdot s(1 \parallel 1) \leqq s(e \parallel f) \end{aligned}$$

In the above, sequential composition binds more tightly than the preclosure operator $\odot$.

In this case, the preclosures coincide with parallel composition, e.g., $\mathtt{a}^* \cdot \mathtt{a} \odot \mathtt{b} = \mathtt{a}^* \cdot \mathtt{a} \parallel \mathtt{b}$. This means that we can simplify the above constraint to

$$\begin{aligned} &\mathtt{a}^* \cdot \mathtt{a} \cdot s(1 \cdot \mathtt{a}^* \parallel \mathtt{b}) \;+\; \mathtt{a}^* \cdot s(\mathtt{a} \cdot \mathtt{a}^* \parallel \mathtt{b}) \;+\; 1 \cdot s(1 \parallel \mathtt{b}) \\ +\; &(\mathtt{a}^* \cdot \mathtt{a} \parallel \mathtt{b}) \cdot s(1 \cdot \mathtt{a}^* \parallel 1) \;+\; (\mathtt{a}^* \parallel \mathtt{b}) \cdot s(\mathtt{a} \cdot \mathtt{a}^* \parallel 1) \;+\; \mathtt{b} \cdot s(1 \parallel 1) \leqq s(e \parallel f) \end{aligned}$$

Similar constraints can be derived on the other variables, which can then be used to compute a least solution s for all variables, and to obtain $s(e \parallel f) = e \otimes f$ in particular.

With the above in place, we are now ready to assert correctness of $e \otimes f$ as a closure of $e \parallel f$.

Lemma 5.36. *If the main induction hypothesis applies to $e \parallel f$, then $e \otimes f$ is a closure of $e \parallel f$.*

Proof. Let $\mathcal{S}$ be the system on states Q obtained for $e \parallel f$, as in Definition 5.33, and let $s : Q \to \mathcal{T}$ be its least solution. We prove the more general claim that $s(g \parallel h)$ is the closure of $g \parallel h$.

To show that $s(g \parallel h) \leqq^{\mathsf{exch}} g \parallel h$, we choose the function $s' : Q \to \mathcal{T}$ by setting $s'(g \parallel h) = g \parallel h$, and claim that s' is a $\langle \equiv^{\mathsf{exch}}, 1\rangle$-solution to the system. To see this, first note $b(g \parallel h) = [g \parallel h \in \mathcal{F}] \leqq g \parallel h = s'(g \parallel h)$ by Lemma 3.53. Furthermore, if $g_0 \nabla_g g'$ and $h_0 \nabla_h h'$, then

$$\begin{aligned}
(g_0 \odot h_0) \cdot s'(g', h') &= (g_0 \odot h_0) \cdot (g' \parallel h') \\
&\equiv^{\mathsf{exch}} (g_0 \parallel h_0) \cdot (g' \parallel h') && \text{(Lemma 5.23)} \\
&\leqq^{\mathsf{exch}} (g_0 \cdot g') \parallel (h_0 \cdot h') && \text{(the exchange law)} \\
&\leqq^{\mathsf{exch}} g \parallel h && \text{(Lemma 5.29(ii))}
\end{aligned}$$

We surmise that for each $g' \in R(e)$ and $h' \in R(f)$, the sr-expression $M(g \parallel h, g' \parallel h') \cdot s'(g \parallel h)$ is included (by $\leqq^{\mathsf{exch}}$) in $g \parallel h = s'(g \parallel h)$. Together with the observation that $b(g \parallel h) \leqq s'(g \parallel h)$, this makes s' a $\langle \equiv^{\mathsf{exch}}, 1\rangle$-solution. Since s is the *least* $\langle \equiv^{\mathsf{exch}}, 1\rangle$-solution, $s(g \parallel h) \leqq^{\mathsf{exch}} s'(g \parallel h) = g \parallel h$.

To show $g \parallel h \leqq s(g \parallel h)$, note that for $g \in R(e)$ and $h \in R(f)$ we obtain by Lemma 5.28 $\ell_1, \dots, \ell_n, \ell'_1, \dots, \ell'_m \in \mathcal{T}$ and $r_1, \dots, r_n, r'_1, \dots, r'_m \in \mathcal{F}$ such that for $1 \leq i \leq n$ and $1 \leq j \leq m$ we have $\ell_i \nabla_h r_i$ and $\ell'_j \nabla_g r'_j$, as well as $g \equiv \ell_1 + \cdots + \ell_n$ and $h \equiv \ell'_1 + \cdots + \ell'_m$. Therefore,

$$g \parallel h \equiv (\ell_1 + \cdots + \ell_n) \parallel (\ell'_1 + \cdots + \ell'_m) \equiv \ell_1 \parallel \ell'_1 + \cdots + \ell_n \parallel \ell'_m \tag{5.1}$$

Take one such sr-expression $\ell_i \parallel \ell'_j$. By Lemma 5.23 and the definition of M, we know that

$$\ell_i \parallel \ell'_j \leqq \ell_i \odot \ell'_j \leqq M(g \parallel h, r_i \parallel r'_j) \tag{5.2}$$

Since $r_i, r'_j \in \mathcal{F}$, we have $r_i \parallel r'_j \in \mathcal{F}$, and hence $1 \leqq s(r_i \parallel r'_j)$. In combination with (5.2) and the fact that s is a solution to the system, we find $\ell_i \parallel \ell'_j \leqq M(g \parallel h, r_i \parallel r'_j) \cdot s(r_i \parallel r'_j) \leqq s(g \parallel h)$. Since each sr-expression on the right hand side of (5.1) is below $s(g \parallel h)$, so is $g \parallel h$, i.e., $g \parallel h \leqq s(g \parallel h)$.

For the second condition on closure, first note that since $s(g \parallel h) \equiv^{\mathsf{exch}} g \parallel h$, we have that $[\![s(g \parallel h)]\!] \subseteq [\![s(g \parallel h)]\!]{\downarrow} = [\![g \parallel h]\!]{\downarrow}$, by Theorem 4.14. It remains to prove the other inclusion, which we do for all $g \parallel h \in Q$ in tandem. To this end, let $U \in [\![g \parallel h]\!]{\downarrow}$, and let $U = U_1 \cdots U_n$ be the sequential factorisation of U. We proceed by induction on n.

- In the base, where $n = 0$, we have that $U = 1$. Hence, by Lemma 3.26, we have that $U \in [\![g \parallel h]\!]$, which means that $g \parallel h \in \mathcal{F}$, and therefore $U \in [\![b(g \parallel h)]\!] \subseteq [\![s(g \parallel h)]\!]$.
- For the inductive step, let $n > 0$ and assume the claim holds for $n-1$. We write $U' = U_2 \cdots U_n$. Since $U_1 \cdot U' = U \in [\![g \parallel h]\!]{\downarrow}$, we find $V \in [\![g]\!]$ and $W \in [\![h]\!]$ such that $U_1 \cdot U' \sqsubseteq V \parallel W$. By Lemma 3.28, we then find pomsets V_0, V_1, W_0, W_1 such that the following hold
$$U_1 \sqsubseteq V_0 \parallel W_0 \qquad U' \sqsubseteq V_1 \parallel W_1 \qquad V_0 \cdot V_1 \sqsubseteq V \qquad W_0 \cdot W_1 \sqsubseteq W$$
By Lemma 5.26, we obtain $\ell_0, \ell_1, r_0, r_1 \in \mathcal{T}$ such that all of the following hold
$$\ell_0 \nabla_g r_0 \qquad \ell_1 \nabla_h r_1 \qquad V_0 \in [\![\ell_0]\!]{\downarrow} \qquad V_1 \in [\![\ell_1]\!]{\downarrow} \qquad W_0 \in [\![r_0]\!]{\downarrow} \qquad W_1 \in [\![r_1]\!]{\downarrow}$$
In that case, since U_1 is a sequential prime, we have that $U_1 \in [\![\ell_0 \odot \ell_1]\!]$ by Lemma 5.23. Furthermore, $U' \in [\![r_0 \parallel r_1]\!]{\downarrow}$, which by induction tells us that $U' \in [\![s(r_0 \parallel r_1)]\!]$. By construction of our sr-system, we have that $(\ell_0 \odot \ell_1) \cdot s(r_0 \parallel r_1) \leqq s(g \parallel h)$; we conclude
$$U = U_1 \cdot U' \in [\![(\ell_0 \odot \ell_1) \cdot s(r_0 \parallel r_1)]\!] \subseteq [\![s(g \parallel h)]\!] \qquad \square$$

This was the last piece of the puzzle necessary to argue computability of closures in general. We conclude by recording the desired reduction and its ramifications for decidability and completeness.

Theorem 5.37. *We can compute a closure $e{\downarrow}$ of $e \in \mathcal{T}$; hence,* exch *strongly reduces to $\emptyset$.*

Corollary 5.38. *Let $e, f \in \mathcal{T}$. The following hold:*

(i) It is decidable whether $[\![e]\!]{\downarrow} = [\![f]\!]{\downarrow}$.

(ii) We have $[\![e]\!]{\downarrow} = [\![f]\!]{\downarrow}$ if and only if $e \equiv^{\mathsf{exch}} f$.

The first of the above is known [BPS17]. The second (along with [LS17]) settles a conjecture by Hoare et al. [HSM+16], namely that semantic equivalence of sr-expressions, when closed w.r.t. subsumption, is completely axiomatised by BKA congruence augmented with the exchange law.

Remark 5.39. If we extend the syntax of series-rational expressions with the operator $(-)^\dagger$ (c.f. Remark 3.39) to arrive at *series-parallel rational expressions*, and augment the semantics $[\![-]\!]$ by setting $[\![e^\dagger]\!] = \{U_1 \parallel \cdots \parallel U_n \ :\ U_1, \dots, U_n \in [\![e]\!]\}$, then a strong reduction of exch to the empty set of hypotheses cannot exist. To see this, first note that there is no upper bound on the nesting between sequential and parallel composition (c.f. Chapter 9) in the pomsets of $[\![\mathsf{a}^\dagger]\!]{\downarrow}$; after all,
$$\mathsf{a},\ \ \mathsf{a} \parallel \mathsf{a},\ \ (\mathsf{a} \parallel \mathsf{a}) \cdot \mathsf{a},\ \ (\mathsf{a} \parallel \mathsf{a}) \cdot \mathsf{a} \parallel \mathsf{a} \ldots \in [\![\mathsf{a}^\dagger]\!]{\downarrow}$$
On the other hand, for any series-parallel rational expression e it can be shown that $[\![e]\!]$ does have such a bound [LS14]. Hence, $[\![\mathsf{a}^\dagger]\!]{\downarrow} \neq [\![e]\!]$ for any series-parallel rational expression e.

5.2 Decomposition

Having reduced the exchange law, one might wonder whether this reduction also applies in the presence of other hypotheses. More specifically, if we have a set of hypotheses H, what are the circumstances under which $H \cup \mathsf{exch}$ can also be reduced to the empty set of hypotheses?

To answer this question, recall that in Section 4.2.3 we developed several tools to compose reductions. In particular, as an instance of Lemma 4.49, we can obtain the following.

Lemma 5.40. *Let H be a set of hypotheses. Then $H \cup \mathsf{exch}$ strongly reduces to $\emptyset$, provided that*

(i) H strongly reduces to $\emptyset$, and

(ii) $H \cup \mathsf{exch}$ factorises into H, exch or exch, H.

Thus, reduction of $H \cup \mathsf{exch}$ largely comes down to whether H itself reduces to the empty set, and whether we can factorise $H \cup \mathsf{exch}$. In this section, we focus on the latter question, in order to derive sufficient conditions that allow for such a factorisation. To get a feeling for which type of hypotheses may be factorised from exch, let us first consider when factorisation fails.

Example 5.41. Let $H = \{\mathtt{a} \leq \mathtt{b} + \mathtt{c}\}$. Then $H \cup \mathsf{exch}$ does *not* factorise into H, exch. To see this, take $L = \{\mathtt{d} \parallel \mathtt{b}, \mathtt{d} \cdot \mathtt{c}\}$. Since $\mathtt{d} \cdot \mathtt{b} \sqsubseteq \mathtt{d} \parallel \mathtt{b}$, we find that $\mathtt{d} \cdot \mathtt{b} \in L^{\mathsf{exch}} \subseteq L^{H \cup \mathsf{exch}}$ (by Corollary 5.6). If we then choose $C = \mathtt{d} \cdot \square$, we find that $C[\llbracket \mathtt{b} + \mathtt{c} \rrbracket] = \{\mathtt{d} \cdot \mathtt{b}, \mathtt{d} \cdot \mathtt{c}\} \subseteq L^{H \cup \mathsf{exch}}$, and hence $\mathtt{d} \cdot \mathtt{a} \in C[\llbracket \mathtt{a} \rrbracket] \subseteq L^{H \cup \mathsf{exch}}$. On the other hand, $\mathtt{d} \cdot \mathtt{a} \notin (L^H)^{\mathsf{exch}}$, thus $(L^H)^{\mathsf{exch}} \neq L^{H \cup \mathsf{exch}}$.

In this example, we see that if $e \leq f \in H$ where $\llbracket f \rrbracket$ contains more than one pomset, then H may not be factorised in front of exch. This is because not all elements of $C[\llbracket f \rrbracket]$ may be present in the language before exch-closure, and $C[\llbracket e \rrbracket]$ can be added only after subsumption closure.

Example 5.42. Let $H = \{\mathtt{a} \leq \mathtt{b} \cdot \mathtt{c}\}$. Then $H \cup \mathsf{exch}$ does *not* factorise into H, exch. To see this, take $L = \{\mathtt{b} \parallel \mathtt{c}\}$. Since $\mathtt{b} \cdot \mathtt{c} \sqsubseteq \mathtt{b} \parallel \mathtt{c}$, we have that $\mathtt{b} \cdot \mathtt{c} \in L^{\mathsf{exch}} \subseteq L^{H \cup \mathsf{exch}}$, and hence $\mathtt{a} \in L^{H \cup \mathsf{exch}}$ as well. Meanwhile, $\mathtt{a} \notin (L^H)^{\mathsf{exch}}$, which shows that $(L^H)^{\mathsf{exch}} \neq L^{H \cup \mathsf{exch}}$.

Thus, if $e \leq f \in H$ where $U \in \llbracket f \rrbracket$ has some non-trivial ordering, then H may also not be factorised in front of exch. The reason here is that the language may contain $V \neq U$ such that $U \sqsubseteq V$, which means that U may be added only after exchange, at which point H can kick in. More generally, the right-hand side of a hypothesis in H should not rely on some causal dependency.

Dually, factorising H after exch may also be impossible, as witnessed by the following.

Example 5.43. Let $H = \{\mathtt{a} \cdot \mathtt{b} \leq \mathtt{c}\}$. Then $H \cup \mathsf{exch}$ does *not* factorise into exch, H. To see this, take $L = \{\mathtt{d} \parallel \mathtt{c}\}$. If we choose $C = \mathtt{d} \parallel \square$, then $C[[\![\mathtt{c}]\!]] \subseteq L$, and hence $\mathtt{d} \parallel \mathtt{a} \cdot \mathtt{b} \in C[[\![\mathtt{a} \cdot \mathtt{b}]\!]] \subseteq L^H$. Since $\mathtt{a} \cdot \mathtt{d} \cdot \mathtt{b} \sqsubseteq \mathtt{d} \parallel \mathtt{a} \cdot \mathtt{b}$, we then know that $\mathtt{a} \cdot \mathtt{d} \cdot \mathtt{b} \in (L^H)^{\mathsf{exch}} \subseteq L^{H \cup \mathsf{exch}}$. On the other hand, $\mathtt{a} \cdot \mathtt{d} \cdot \mathtt{b} \not\in (L^{\mathsf{exch}})^H$, and therefore $(L^{\mathsf{exch}})^H \neq L^{H \cup \mathsf{exch}}$.

Here, we see that if the left-hand side $e \leq f \in H$ contains some non-trivial ordering, then first closing w.r.t. exch followed by closing w.r.t. this hypothesis may mean that we miss out on some pomsets. After all, if can close w.r.t. $e \leq f$ first, then the events internal to e may be interleaved differently with other events. In the example, this is witnessed by $\mathtt{a}$ being ordered before $\mathtt{d}$, while $\mathtt{b}$ is placed after $\mathtt{d}$, whereas first closing L w.r.t. exch means that $\mathtt{c}$ is interleaved as a whole, after which it can be substituted by $\mathtt{a} \cdot \mathtt{b}$, which means that both letters end up before or after $\mathtt{d}$.

Remark 5.44. A set of hypotheses of the form $H = \{\mathtt{a} + \mathtt{b} \leq \mathtt{c}\}$ would be the next format to test for factorisation. However, one can show that this set is equivalent (by way of mutual implication) to the set $H' = \{\mathtt{a} \leq \mathtt{c}, \mathtt{b} \leq \mathtt{c}\}$, which, as we shall see, can be factorised out of $H' \cup \mathsf{exch}$.

In summary, we see that if we want to factorise H before exch, then the right-hand side of hypotheses in H should not contain more than one pomset, nor should it contain pomsets with non-trivial order; this leaves us with right-hand sides that contain at most one pomset, which is primitive or empty. Similarly, if we want to factorise H after exch, then the left-hand side of hypotheses in H should not contain pomsets with non-trivial order; hypotheses whose left-hand side contains more than one pomset can, in some cases, be converted into several hypotheses where this is not the case. We thus arrive at the same restriction for left-hand sides.

Definition 5.45 (Simple hypotheses)**.** Let $e \leq f$ be a hypothesis. We say that $e \leq f$ is *left-simple* if $e = 1$ or $e = \mathtt{a}$ for some $\mathtt{a} \in \mathcal{T}$; similarly, we say that $e \leq f$ is *right-simple* if $f = 1$ or $f = \mathtt{a}$ for some $\mathtt{a} \in \mathcal{T}$. A set of hypotheses is called left-simple (resp. right-simple) if each of its elements is.

We now focus our attention on the case where H is left-simple, and we try to factorise it after exch, as well as the case where where H is right-simple, and we try to factorise it before exch. Since exch-closure is the same as $\sqsubseteq$-downward closure, it pays to think about how the contexts involved in H-closure relate to subsumption. In particular, suppose that we are computing the closure of a language L w.r.t. $H \cup \mathsf{exch}$, and we have added U to this closure because $V \in L^{H \cup \mathsf{exch}}$ and $U \sqsubseteq V$. Next, we note that $C \in \mathsf{PC}^{\mathsf{sp}}$ is such that $C[\mathtt{a}] = U$, and because $e \leq \mathtt{a} \in H$, we are about to add $C[[\![e]\!]]$ to the closure as well. What we could have done instead was find a context $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C'[\mathtt{a}] = V$ and $C \sqsubseteq C'$; then, we could have added $C'[[\![e]\!]]$ first, followed by $C[[\![e]\!]]$.

Finding such a C' is not hard: take the events in C and superimpose the order of V, where the event labelled $\square$ plays the role of the substituted event. An example is depicted in Figure 5.2a. For this approach to work, we need to formalise how to work with the partial order implicit in an sp-pomset context; to this end, we generalise sp-pomset contexts, as follows:

Definition 5.46. We define the set of *pomset contexts*, denoted PC, as containing all pomsets over $\Sigma \cup \{\square\}$ that have *exactly one* node labelled by $\square$. When $C \in \mathsf{PC}$ with $C = [\mathbf{c}]$, we denote this node by $s_\square \in S_\mathbf{c}$. When furthermore $\mathbf{u}$ is a labelled poset over Σ, we write $\mathbf{c}[\mathbf{u}]$ for the labelled poset where $S_{\mathbf{c}[\mathbf{u}]} = S_\mathbf{u} \cup S_\mathbf{c} \setminus \{s_\square\}$ and $\lambda_{\mathbf{c}[\mathbf{u}]}(s)$ is given by $\lambda_\mathbf{c}(s)$ if $s \in S_\mathbf{c} \setminus \{s_\square\}$, and $\lambda_\mathbf{u}(s)$ when $s \in S_\mathbf{u}$. Lastly, $\leq_{\mathbf{c}[\mathbf{u}]}$ is the smallest relation on $S_{\mathbf{c}[\mathbf{u}]}$ satisfying the following rules[1]

$$\frac{s \leq_\mathbf{u} s'}{s \leq_{\mathbf{c}[\mathbf{u}]} s'} \qquad \frac{s \leq_\mathbf{c} s'}{s \leq_{\mathbf{c}[\mathbf{u}]} s'} \qquad \frac{s_\square \leq_\mathbf{c} s \quad s' \in S_\mathbf{u}}{s' \leq_{\mathbf{c}[\mathbf{u}]} s} \qquad \frac{s' \in S_\mathbf{u} \quad s \leq_\mathbf{c} s_\square}{s \leq_{\mathbf{c}[\mathbf{u}]} s'}$$

The sp-pomset contexts are precisely the pomset contexts that are series-parallel. Furthermore, the substitution on the level of labelled posets that represent a pomset context relates to substitution.

Lemma 5.47. *Let $C \in \mathsf{PC}$; the following hold:*

(i) $C \in \mathsf{PC}^{\mathsf{sp}}$ if and only if C is series-parallel, and

(ii) if $C = [\mathbf{c}] \in \mathsf{PC}^{\mathsf{sp}}$ and $U = [\mathbf{u}] \in \mathsf{Pom}$, then $C[U] = [\mathbf{c}[\mathbf{u}]]$.

The above justifies extending our notation for plugging holds in contexts to pomset contexts in general, i.e., when $C = [\mathbf{c}] \in \mathsf{PC}$ and $U = [\mathbf{u}] \in \mathsf{Pom}$, we write $C[U]$ for the pomset represented by the labelled poset $\mathbf{c}[\mathbf{u}]$; in the special case where $C \in \mathsf{PC}^{\mathsf{sp}}$, this is the same as the old definition of $C[U]$. This then gives us the flexibility to work out our earlier idea, as follows.

Lemma 5.48. *Let $C \in \mathsf{PC}$, $V \in \mathsf{Pom}$ and $\mathsf{a} \in \Sigma$. The following hold:*

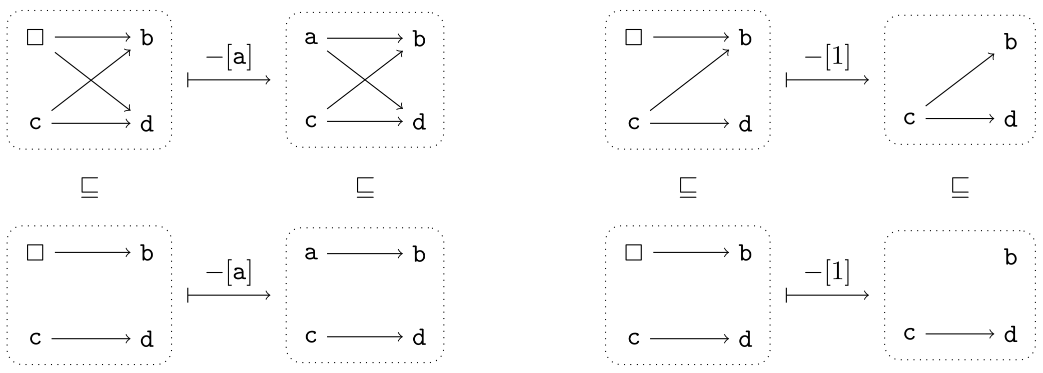
(i) If $C[\mathsf{a}] \sqsubseteq V$, then we can construct $C' \in \mathsf{PC}$ s.t. $C \sqsubseteq C'$ and $C'[\mathsf{a}] = V$.

(ii) If $V \sqsubseteq C[\mathsf{a}]$, then we can construct $C' \in \mathsf{PC}$ s.t. $C' \sqsubseteq C$ and $C'[\mathsf{a}] = V$.

Moreover, if $V \in \mathsf{SP}$, then the C' constructed in each of the above is series-parallel.

Conceptually, the same construction would also work if instead of a primitive pomset a, we plug in the empty pomset. There is, however, one wrinkle: the context obtained using this method may not be series-parallel, that is to say, it may contain an N-shape, such as depicted in Figure 5.2b. This is a problem, because closure w.r.t. a set of hypotheses is defined using series-parallel contexts.

[1]It can easily be shown that these rules make $\leq_{\mathbf{c}[\mathbf{u}]}$ a partial order.



(a) On the upper left, we have a context C and on the upper right the pomset $C[\mathtt{a}] = U$, which is subsumed by the pomset V on the lower right. The context C' on the lower left subsumes C, such that $C'[\mathtt{a}]$ yields V. Dually, we can have the context C on the lower left, and $C[\mathtt{a}]$ on the lower right. The pomset V subsumed by $C[\mathtt{a}]$ is on the upper right. The context C' on the upper left is subsumed by C such that $C'[\mathtt{a}] = V$.

(b) A version of Lemma 5.48 where we substitute by the empty pomset may produce pomset contexts that are not series-parallel. In the above, we have the context C on the lower left, and $C[1]$ on the lower left, which subsumes V on the upper right. If we take the ordering of V and impose it on C, we obtain the context C' on the upper left, which is subsumed by C, and furthermore $C'[1] = V$. However, C' is not series-parallel.

Figure 5.2: Transposition of subsumption across contexts.

However, we can add or remove order to ensure series-parallelism, while preserving the pomset obtained. For instance, in Figure 5.2b we can connect the event labelled $\square$ to the one labelled $\mathtt{d}$ on the upper left; this gives us a context that is still subsumed by the context on the lower left, while plugging in 1 still yields the pomset on the upper right. In general, we find the following.

Lemma 5.49. *Let $C \in \mathsf{PC}$ and $U \in \mathsf{SP}$ such that $C[1] = U$. The following hold:*

(i) We can construct a $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C'[1] = U$ and $C \sqsubseteq C'$.

(ii) We can construct a $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C'[1] = U$ and $C' \sqsubseteq C$.

With the above in hand, we can obtain an analogue of Lemma 5.48.

Lemma 5.50. *Let $C \in \mathsf{PC}$ and $V \in \mathsf{Pom}$. The following hold:*

(i) If $C[1] \sqsubseteq V$, then we can construct $C' \in \mathsf{PC}$ such that $C \sqsubseteq C'$ and $C'[1] = V$.

(ii) If $V \sqsubseteq C[1]$, then we can construct $C' \in \mathsf{PC}$ such that $C' \sqsubseteq C$ and $C'[1] = V$.

Moreover, if $V \in \mathsf{SP}$, then the C' constructed in each of the above is series-parallel.

In the proof of the above, we can simply argue the first two claims using the same technique as Lemma 5.48; the latter is a consequence of Lemma 5.49.

Using Lemmas 5.48 and 5.50, we can then show that exch-closure followed by H-closure with H a left-simple set of hypotheses H yields a language that is still exch-closed. Dually, H-closure where H is a right-simple set of hypotheses, followed by exch-closure yields an H-closed language.

Lemma 5.51. *Let H be a set of hypotheses, let $L \subseteq \mathsf{SP}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$. The following hold:*

(i) If H is right-simple and $e \leq f \in H$ such that $C[\llbracket f \rrbracket] \subseteq (L^H)^{\mathsf{exch}}$, then $C[\llbracket e \rrbracket] \subseteq (L^H)^{\mathsf{exch}}$.

(ii) If H is left-simple and $e \leq f \in \mathsf{exch}$ such that $C[\llbracket f \rrbracket] \subseteq (L^{\mathsf{exch}})^H$, then $C[\llbracket e \rrbracket] \subseteq (L^{\mathsf{exch}})^H$.

This is the final piece of the puzzle necessary to show that simple hypotheses can be factored out when encountered in conjunction with the exchange law, with left-simple hypotheses appearing after exchange, and right-simple hypotheses appearing before exchange.

Theorem 5.52. *Let H be a set of hypotheses. The following hold.*

(i) If H is right-simple, then $H \cup \mathsf{exch}$ factorises into H, exch.

(ii) If H is left-simple, then $H \cup \mathsf{exch}$ factorises into exch, H.

We can then reap the benefits of our factorisation theorem, by combining it with Lemma 5.40 to obtain the following statement about strong reduction of sets of hypotheses that include exch.

Corollary 5.53. *Suppose that H is left-simple or right-simple, and that H strongly reduces to $\emptyset$. Then $H \cup \mathsf{exch}$ strongly reduces to $\emptyset$ as well.*

In the above, we require that all hypotheses in H are left-simple, or that all are right-simple. This may not always be the case; it could be that some hypotheses are left-simple, and others are right-simple. From Lemma 4.50, we learned that if we have a set of hypotheses consisting of several smaller sets of hypotheses, then we can factorise this larger set as well, provided that the union of each of the smaller sets can be factorised. Since left-simple hypotheses factorise after exchange, and right-simple hypotheses factorise before exchange, it then suffices to verify that the left-simple hypotheses factorise after the right-simple ones. We then obtain the following.

Corollary 5.54. *Let H_ℓ be left-simple, and let H_r be a right-simple set of hypotheses. If $H_\ell \cup H_r$ factorises into H_ℓ, H_r, then $H_\ell \cup \mathsf{exch} \cup H_r$ factorises into $H_\ell, \mathsf{exch}, H_r$.*

If the left-simple and right-simple hypotheses themselves again reduce to the empty set of hypotheses, then the union of these with the exchange law also reduces to the empty set.

Corollary 5.55. *Let H_ℓ be a left-simple set of hypotheses, and let H_r be a right-simple set of hypotheses. If both of these strongly reduce to $\emptyset$, and $H_\ell \cup H_r$ factorises into H_ℓ, H_r, then $H_\ell \cup \mathsf{exch} \cup H_r$ strongly reduces to $\emptyset$.*

Summary of this chapter The possibility of interleaving is commonly represented by adding the exchange law to the axioms of bi-Kleene algebra. We showed that the exchange law can be regarded as a set of hypotheses, exch, and that closure w.r.t. these hypotheses coincides with downward closure w.r.t. subsumption. We then went on to show that exch can be reduced to the empty set of hypotheses. As a consequence, we derived completeness and decidability for series-rational expressions with respect to subsumption closure. We also showed that, under certain conditions, the exchange law can be factorised out of a larger set of hypotheses.

5.A Proofs to reduce the exchange law

Lemma 5.5 [Gis88]. *Let $\sqsubseteq^{\mathsf{sp}}$ be $\sqsubseteq$ restricted to SP. Then $\sqsubseteq^{\mathsf{sp}}$ is the smallest precongruence satisfying the exchange law, i.e., such that for all $U, V, W, X \in \mathsf{SP}$, it holds that*

$$(U \parallel V) \cdot (W \parallel X) \sqsubseteq^{\mathsf{sp}} (U \cdot W) \parallel (V \cdot X)$$

Proof. Let $\leqslant$ be the smallest precongruence satisfying the above. To show that $\leqslant$ is included in $\sqsubseteq^{\mathsf{sp}}$, we need to show that $\sqsubseteq^{\mathsf{sp}}$ satisfies the rules that define $\leqslant$. To show that $\sqsubseteq^{\mathsf{sp}}$ is a precongruence, we first note that it is a preorder (c.f. Lemma 3.B.1). To show compatibility with the operators, let $U, U', V, V' \in \mathsf{SP}$ be such that $U = [\mathbf{u}]$, $U' = [\mathbf{u}']$, $V = [\mathbf{v}]$ and $V' = [\mathbf{v}']$, and let $U \sqsubseteq^{\mathsf{sp}} U'$ and $V \sqsubseteq^{\mathsf{sp}} V'$ be witnessed by $h : S_{\mathbf{u}'} \to S_{\mathbf{u}}$ and $g : S_{\mathbf{v}'} \to S_{\mathbf{v}}$ respectively. We then define $f : S_{\mathbf{u}'} \cup S_{\mathbf{v}'} \to S_{\mathbf{u}} \cup S_{\mathbf{v}}$ by setting $f(x) = h(x)$ when $x \in S_{\mathbf{u}'}$ and $f(x) = g(x)$ when $x \in S_{\mathbf{v}'}$.

- To show $U \parallel V \sqsubseteq^{\mathsf{sp}} U' \parallel V'$, suppose $x \leq_{\mathbf{u}' \parallel \mathbf{v}'} x'$. Then either $x \leq_{\mathbf{u}'} x'$ or $x \leq_{\mathbf{v}'} x'$; in the first case, $f(x) \leq_{\mathbf{u}} f(x')$, whence $f(x) \leq_{\mathbf{u} \parallel \mathbf{v}} f(x')$ — the second case can be treated similarly.
- To show $U \cdot V \sqsubseteq^{\mathsf{sp}} U' \cdot V'$, suppose $x \leq_{\mathbf{u}' \cdot \mathbf{v}'} x$. If $x \leq_{\mathbf{u}} x'$ or $x \leq_{\mathbf{v}} x'$, then we find that $f(x) \leq_{\mathbf{u} \cdot \mathbf{v}} f(x')$ analogously to the case for parallel composition. Otherwise, if $x \in S_{\mathbf{u}'}$ and $x' \in S_{\mathbf{v}'}$, then $f(x) \in S_{\mathbf{u}}$ and $x' \in S_{\mathbf{v}}$, which means that $f(x) \leq_{\mathbf{u} \cdot \mathbf{v}} f(x')$.

It remains to show that $\sqsubseteq^{\mathsf{sp}}$ is included in $\leqslant$. To this end, suppose $U \sqsubseteq^{\mathsf{sp}} V$. We proceed by induction on the size of U (which is the same size as V). In the base, U is empty or primitive, in which case $U = V$ by Lemma 3.26, and thus $U \leqslant V$. For the inductive step, there are three cases.

- If U is parallel, then $U = U_0 \parallel U_1$ for non-empty $U_0, U_1 \in \mathsf{SP}$. By Lemma 3.27, we find that $V = V_0 \parallel V_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$. Furthermore, Theorem 3.22 tells us that since V is N-free, V_0 and V_1 are also N-free, and hence $V_0, V_1 \in \mathsf{SP}$. By induction, we then find that $U_0 \leqslant V_0$ and $U_1 \leqslant V_1$, from which we conclude that $U = U_0 \parallel U_1 \leqslant V_0 \parallel V_1 = V$.
- If V is sequential, then $V = V_0 \cdot V_1$ for non-empty $V_0, V_1 \in \mathsf{SP}$. By Lemma 3.27, we find that $U = U_0 \cdot U_1$ such that $U_0 \sqsubseteq V_0$ and $U_1 \sqsubseteq V_1$. Furthermore, Theorem 3.22 tells us that since V is N-free, so are U_0 and U_1, and hence $U_0, U_1 \in \mathsf{SP}$. By induction, we find that $U_0 \leqslant V_0$ and $U_1 \leqslant V_1$, which means that $U = U_0 \cdot U_1 \leqslant V_0 \cdot V_1 = V$.
- In the final case, U is neither empty, nor, primitive, nor sequential, which means it must be parallel. Similarly, V is neither empty, nor sequential, nor parallel, which means that it must be sequential. We thus write $U = U_0 \cdot U_1$ and $V = V_0 \parallel V_1$ for $U_0, U_1, V_0, V_1 \in \mathsf{SP}$. By Lemma 4.49, we find $W_0, W_1, X_0, X_1 \in \mathsf{SP}$ such that

  $$W_0 \cdot W_1 \sqsubseteq V_0 \qquad X_0 \cdot X_1 \sqsubseteq V_1 \qquad U_0 \sqsubseteq W_0 \parallel X_0 \qquad U_1 \sqsubseteq W_1 \parallel X_1$$

  All of these pomsets are strictly smaller than U, and hence by induction we find that

  $$W_0 \cdot W_1 \leqslant V_0 \qquad X_0 \cdot X_1 \leqslant V_1 \qquad U_0 \leqslant W_0 \parallel X_0 \qquad U_1 \leqslant W_1 \parallel X_1$$

  This then allows us to derive that

  $$U = U_0 \cdot U_1 \leqslant (W_0 \parallel X_0) \cdot (W_1 \parallel X_1) \leqslant (W_0 \cdot W_1) \parallel (X_0 \cdot X_1) \leqslant V_0 \parallel V_1 = V \qquad \square$$

Lemma 5.11 (c.f. [Gis88, Theorems 5.2 and 5.4])**.** *Let $L, K \subseteq \mathsf{SP}$ and $\mathtt{a} \in \Sigma$. The following hold:*

$$\{1\}{\downarrow} = \{1\} \qquad \{\mathtt{a}\}{\downarrow} = \{\mathtt{a}\} \qquad (L \cup K){\downarrow} = L{\downarrow} \cup K{\downarrow} \qquad (L \cdot K){\downarrow} = L{\downarrow} \cdot K{\downarrow} \qquad L^*{\downarrow} = (L{\downarrow})^*$$

Proof. The inclusion from right to left for the first two equalities holds by definition of closure. For the last three equalities, this inclusion follows from Lemma 4.13 and Corollary 5.6. It remains to prove the inclusions from left to right, which we do in the order of the claims.

- If $U \in \{1\}{\downarrow}$, then $U \sqsubseteq 1$, which means that $U = 1$ by Lemma 3.26, which means that $U \in \{1\}$.
- If $U \in \{\mathtt{a}\}{\downarrow}$, then $U \sqsubseteq \mathtt{a}$, which means that $U = \mathtt{a}$ by Lemma 3.26, which means that $U \in \{\mathtt{a}\}$.
- If $U \in (L \cup K){\downarrow}$, then $U \sqsubseteq V$ for some $V \in L \cup K$. If $U \in L$, then $U \in L{\downarrow}$, which means that $U \in L{\downarrow} \cup K{\downarrow}$. The case where $U \in K$ can be treated similarly.
- If $U \in (L \cdot K){\downarrow}$, then $U \sqsubseteq V \cdot W$ where $V \in L$ and $W \in K$. By Lemma 3.27, we find that $U = X \cdot Y$ where $X \sqsubseteq V$ and $W \sqsubseteq K$. It then follows that $X \in L{\downarrow}$ and $W \in K{\downarrow}$, which means that $V \in L{\downarrow} \cdot K{\downarrow}$.

- If $U \in L^*{\downarrow}$, then $U \sqsubseteq V_1 \cdots V_n$ for $V_1, \dots, V_n \in L$. By Lemma 3.27, we find that $U = U_1 \cdots U_n$ such that for $1 \leq i \leq n$ we have $U_i \sqsubseteq V_i$. From this, we find that for $1 \leq i \leq n$ we have $U_i \in L{\downarrow}$, and hence $U \in (L{\downarrow})^*$. □

Lemma 5.18. *Let $e \in \mathcal{T}$. If $V, W \in \mathsf{SP}$ are non-empty and $V \parallel W \in [\![e]\!]$, then there exist $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\Delta_e} r$ such that $V \in [\![\ell]\!]$ and $W \in [\![r]\!]$.*

Proof. The proof proceeds by induction on e. In the base, there are two cases.

- If $e = 1$, then $V \parallel W \in [\![e]\!]$ entails $V \parallel W = 1$. This means that $V = W = 1$. Since $1 \mathrel{\Delta_e} 1$ by definition of Δ_e, the claim follows when we choose $\ell = r = 1$.
- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $V \parallel W \in [\![e]\!]$ entails $V \parallel W = \mathtt{a}$. Hence, either $V = 1$ and $W = \mathtt{a}$, or $V = \mathtt{a}$ and $W = 1$. In the former case, we can choose $\ell = 1$ and $r = \mathtt{a}$, while in the latter case we can choose $\ell = \mathtt{a}$ and $r = 1$. In either case, the claim follows.

For the inductive step, there are four cases to consider.

- If $e = e_0 + e_1$, then $U_0 \parallel U_1 \in [\![e_i]\!]$ for some $i \in \{0, 1\}$. By induction, we find $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\Delta_{e_i}} r$ such that $V \in [\![\ell]\!]$ and $W \in [\![r]\!]$. Since this implies that $\ell \mathrel{\Delta_e} r$, the claim follows.
- If $e = e_0 \cdot e_1$, then there exist pomsets U_0, U_1 such that $V \parallel W = U_0 \cdot U_1$, and $U_i \in [\![e_i]\!]$ for all $i \in \{0, 1\}$. Since a pomset cannot be both sequential and parallel, it must be the case that $U_i = 1$ for some $i \in \{0, 1\}$, meaning that $V \parallel W = U_0 \cdot U_1 = U_{1-i} \in [\![e_{1-i}]\!]$ for this i. By induction, we find $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\Delta_{e_{1-i}}} r$, and $V \in [\![\ell]\!]$ as well as $W \in [\![r]\!]$. Since $U_i = 1 \in [\![e_i]\!]$, we have that $e_i \in \mathcal{F}$ by Lemma 3.53, and thus $\ell \mathrel{\Delta_e} r$.
- If $e = e_0 \parallel e_1$, then there exist pomsets U_0, U_1 such that $V \parallel W = U_0 \parallel U_1$, and $U_i \in [\![e_i]\!]$ for all $i \in \{0, 1\}$. By Lemma 3.A.2, we find pomsets V_0, V_1, W_0, W_1 such that $V = V_0 \parallel V_1$, $W = W_0 \parallel W_1$, and $U_i = V_i \parallel W_i$ for $i \in \{0, 1\}$. This gives us three subcases.
  - If V_0 is empty, but V_1 and W_1 are not, then $V = V_1$ and $U_0 = W_0$. By induction we obtain $\ell', r' \in \mathcal{T}$ s.t. $\ell' \mathrel{\Delta_{e_1}} r'$, with $W_1 \in [\![\ell']\!]$ and $V_1 \in [\![r']\!]$. We choose $\ell = e_0 \parallel \ell'$ and $r = r'$ to find $\ell \mathrel{\Delta_e} r$, with $W = W_0 \parallel W_1 = U_0 \parallel W_1 \in [\![e_0 \parallel \ell']\!]$ and $V = V_1 \in [\![r']\!] = [\![r]\!]$. The cases where (a) V_1 is empty but V_0 and W_0 are not, (b) W_0 is empty, but W_1 and V_1 are not, and (c) W_1 is empty, but W_0 and V_0 are not can be handled similarly.
  - If V_0 and W_1 are empty, while V_1 and W_0 are not, then $V = V_1$ and $W = W_0$. We can then choose $\ell = e_0$ and $r = e_1$ to find that $V = V_1 \in [\![\ell]\!]$ and $W = W_0 \in [\![r]\!]$, while $\ell \mathrel{\Delta_e} r$. The case where V_1 and W_0 are empty, while V_0 and W_1 are not, is similar.

  - If none of V_0, V_1, W_0, W_1 is empty, then we proceed as follows. For $i \in \{0,1\}$, we find by induction $\ell_i, r_i \in \mathcal{T}$ with $\ell_i \mathrel{\Delta_{e_i}} r_i$ such that $V_i \in [\![\ell_i]\!]$ and $W_i \in [\![r_i]\!]$. We then choose $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$. Since $V = V_0 \parallel V_1$, it follows that $V \in [\![\ell]\!]$, and similarly we find that $W \in [\![r]\!]$. Since $\ell \mathrel{\Delta_e} r$, the claim follows.

  Note that this covers all of the cases, since no more than two of V_0, V_1, W_0, W_1 can be empty at the same time, and V_0 and V_1 cannot both be empty, as can W_0 and W_1.

- If $e = e_0^*$, then there exist $U_1, \dots, U_n \in [\![e_0]\!]$ such that $V \parallel W = U_1 \cdots U_n$. We can assume without loss of generality that, for $0 \leq i < n$, we have $U_i \neq 1$. Since both V and W are non-empty, it must be the case that $n = 1$ — otherwise $U_j = 1$ for some $1 \leq i \leq n$, which would contradict the above. Since $V \parallel W = U_0 \in [\![e_0]\!]$, we find by induction $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\Delta_{e_0}} r$ such that $V \in [\![\ell]\!]$ and $W \in [\![r]\!]$. The claim then follows by the fact that $\ell \mathrel{\Delta_e} r$. □

Lemma 5.19. *Let $e \in \mathcal{T}$. The following hold:*

(i) There are finitely many $\ell, r \in \mathcal{T}$ such that $\ell \mathrel{\Delta_e} r$.

(ii) If $\ell \mathrel{\Delta_e} r$, then $\ell \parallel r \leqq e$.

(iii) If $\ell \mathrel{\Delta_e} r$, then $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r) < \mathsf{d}_{\parallel}(e)$.

Proof. We prove the claims in the order given.

(i) The proof proceeds by induction on e, showing that we derive a constraint on ℓ and r that is satisfied by only finitely many sr-expressions. In the base, where $e \in \{0,1\} \cup \Sigma$, the claim holds immediately: since only the first rule applies, it must be the case that $\{\!|\ell, r|\!\} = \{\!|1, e|\!\}$. For the inductive step, suppose that $\ell \mathrel{\Delta_e} r$; one of seven cases must apply.

  - $e = e_0 + e_1$, with either $\ell \mathrel{\Delta_{e_0}} r$, or $\ell \mathrel{\Delta_{e_1}} r$.
  - $e = e_0 \cdot e_1$, with an $i \in \{0,1\}$ such that $\ell \mathrel{\Delta_{e_i}} r$ and $e_{1-i} \in \mathcal{F}$.
  - $e = e_0 \parallel e_1$, with $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$, such that $\ell_i \mathrel{\Delta_{e_i}} r_i$ for all $i \in \{0,1\}$.
  - $e = e_0 \parallel r$, with $\ell = \ell' \parallel r'$ and $\ell' \mathrel{\Delta_{e_0}} r'$.
  - $e = \ell \parallel e_1$, with $r = \ell' \parallel r'$ and $\ell' \mathrel{\Delta_{e_1}} r'$.
  - $e = \ell \parallel r$, with no further constraints on ℓ and r.
  - $e = e_0^*$, with $\ell \mathrel{\Delta_{e_0}} r$.

  In all of these, there are finitely many $\ell, r \in \mathcal{T}$ that satisfy the derived restrictions — by induction in all but the penultimate case, where ℓ and r are uniquely determined.

(ii) We proceed by induction on the construction of Δ. In the base, $\ell \mathrel{\Delta_e} r$ because $e = \ell \parallel r$, in which case the claim holds vacuously. For the inductive step, there are five cases to consider.

- If $e = e_0 \parallel e_1$ with $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$ such that $\ell_i \mathrel{\Delta_{e_i}} r_i$ for $i \in \{0, 1\}$, then by induction we have that $\ell_i \parallel r_i \leqq e_i$. In total, we can derive that

$$\ell \parallel r = \ell_0 \parallel \ell_1 \parallel r_0 \parallel r_1 = \ell_0 \parallel r_0 \parallel \ell_1 \parallel r_1 \leqq e_0 \parallel e_1 = e$$

- If $e = e_0 \parallel e_1$ with $r = r' \parallel e_1$ and $\ell \mathrel{\Delta_{e_0}} r'$, then by induction we have that $\ell \parallel r' \leqq e_0$. We can then derive that $\ell \parallel r = \ell \parallel r' \parallel e_1 \leqq e_0 \parallel e_1 = e$. The case where $e = e_0 \parallel e_1$ with $\ell = e_0 \parallel \ell'$ and $\ell' \mathrel{\Delta_{e_1}} r$ can be treated similarly.
- If $e = e_0 + e_1$ and $\ell \mathrel{\Delta_{e_0}} r$, then by induction we have that $\ell \parallel r \leqq e_0 \leqq e_0 + e_1 = e$. The case where $e = e_0 + e_1$ and $\ell \mathrel{\Delta_{e_1}} r$ can be treated similarly.
- If $e = e_0 \cdot e_1$ and $\ell \mathrel{\Delta_{e_0}} r$ with $e_1 \in \mathcal{F}$, then by induction we have that $\ell \parallel r \leqq e_0$. Since $1 \leqq e_1$ by Lemma 3.53, we can then derive that $\ell \parallel r \leqq e_0 \equiv e_0 \cdot 1 \leqq e_0 \cdot e_1 = e$. The case where $e = e_0 \cdot e_1$ and $\ell \mathrel{\Delta_{e_1}} r$ with $e_0 \in \mathcal{F}$ can be treated similarly.
- If $e = e_0^*$ and $\ell \mathrel{\Delta_{e_0}} r$, then by induction we have that $\ell \parallel r \leqq e_0 \leqq e_0 \cdot e_0^* \leqq e_0^*$.

(iii) We proceed by induction on the construction of Δ. In the base, where $e = \ell \parallel r$, we have that $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r) < \max(\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r)) + 1 = \mathsf{d}_\parallel(e)$. For the inductive step, there are five cases.

- If $e = e_0 \parallel e_1$ with $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$ such that $\ell_i \mathrel{\Delta_{e_i}} r_i$ for $i \in \{0, 1\}$, then by induction we have that $\mathsf{d}_\parallel(\ell_i), \mathsf{d}_\parallel(r_i) < \mathsf{d}_\parallel(e_i)$. We can then derive that

$$\mathsf{d}_\parallel(\ell) = \max(\mathsf{d}_\parallel(\ell_0), \mathsf{d}_\parallel(\ell_1)) + 1 < \max(\mathsf{d}_\parallel(e_0), \mathsf{d}_\parallel(e_1)) + 1 = \mathsf{d}_\parallel(e)$$

  and similarly for $\mathsf{d}_\parallel(r)$.
- If $e = e_0 \parallel e_1$ with $r = r' \parallel e_1$ and $\ell \mathrel{\Delta_{e_0}} r'$, then by induction we have that $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r') < \mathsf{d}_\parallel(e_0) \leq \mathsf{d}_\parallel(e)$. We can then derive $\mathsf{d}_\parallel(r) = \max(\mathsf{d}_\parallel(r'), \mathsf{d}_\parallel(e_1)) + 1 < \max(\mathsf{d}_\parallel(e_0), \mathsf{d}_\parallel(e_1)) + 1 = \mathsf{d}_\parallel(e)$. The case where $e = e_0 \parallel e_1$ with $\ell = e_0 \parallel \ell'$ and $\ell' \mathrel{\Delta_{e_1}} r$ can be treated similarly.
- If $e = e_0 + e_1$ and $\ell \mathrel{\Delta_{e_0}} r$, then by induction we have that $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r) < \mathsf{d}_\parallel(e_0)$. Since $\mathsf{d}_\parallel(e_0) \leq \mathsf{d}_\parallel(e)$, the claim then follows. The case where $e = e_0 + e_1$ and $\ell \mathrel{\Delta_{e_1}} r$ is similar.
- If $e = e_0 \cdot e_1$ with $\ell \mathrel{\Delta_{e_0}} r$ and $e_1 \in \mathcal{F}$, then by induction $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r) < \mathsf{d}_\parallel(e_0)$. Since $\mathsf{d}_\parallel(e_0) \leq \mathsf{d}_\parallel(e)$, the claim then follows. The case where $e = e_0 \cdot e_1$ with $\ell \mathrel{\Delta_{e_1}} r$ is similar.
- If $e = e_0^*$ and $\ell \mathrel{\Delta_{e_0}} r$, then by induction we have that $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r) < \mathsf{d}_\parallel(e_0)$. Since $\mathsf{d}_\parallel(e_0) = \mathsf{d}_\parallel(e)$, the claim then follows. $\square$

Lemma 5.26. *Let $e \in \mathcal{T}$, and let V and W be pomsets such that $V \cdot W \in [\![e]\!]{\downarrow}$. Then there exist $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\nabla_e} r$ such that $V \in [\![\ell]\!]{\downarrow}$ and $W \in [\![r]\!]{\downarrow}$.*

Proof. The proof proceeds by induction on e. In the base, we can discount the case where $e = 0$, for then the claim holds vacuously. This leaves us two cases.

- If $e = 1$, then $V \cdot W = 1$, and hence $V = W = 1$. Since $1 \mathrel{\nabla_e} 1$, we can just choose $\ell = r = 1$.
- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $V \cdot W = \mathtt{a}$, and hence either $V = 1$ and $W = \mathtt{a}$, or $V = \mathtt{a}$ and $W = 1$. In the former case, choose $\ell = \mathtt{a}$ and $r = 1$; the latter case is similar.

For the inductive step, there are four cases to consider.

- If $e = e_0 + e_1$, then $V \cdot W \in [\![e_i]\!]{\downarrow}$ for some $i \in \{0, 1\}$. By induction, we find $\ell, r \in \mathcal{T}$ with $\ell \mathrel{\nabla_{e_i}} r$ such that $V \in [\![\ell]\!]{\downarrow}$ and $W \in [\![r]\!]{\downarrow}$. Since $\ell \mathrel{\nabla_e} r$ in this case, the claim follows.
- If $e = e_0 \cdot e_1$, then there exist $U_0 \in [\![e_0]\!]{\downarrow}$ and $U_1 \in [\![e_1]\!]{\downarrow}$ such that $V \cdot W = U_0 \cdot U_1$. By Lemma 3.A.1, we find a series-parallel pomset X such that either $V = U_0 \cdot X$ and $X \cdot W = U_1$, or $V \cdot X = U_0$ and $W = X \cdot U_1$. In the former case, $X \cdot W \in [\![e_1]\!]{\downarrow}$, and thus by induction we find $\ell', r \in \mathcal{T}$ with $\ell' \mathrel{\nabla_{e_1}} r$ such that $X \in [\![\ell']\!]{\downarrow}$ and $W \in [\![r]\!]{\downarrow}$. We then choose $\ell = e_0 \cdot \ell'$ to find that $\ell \mathrel{\nabla_e} r$, as well as $V = U_0 \cdot X \in [\![e_0]\!]{\downarrow} \cdot [\![\ell']\!]{\downarrow} = [\![\ell]\!]{\downarrow}$ and thus $V \in [\![\ell]\!]{\downarrow}$. The latter case can be treated similarly; here, we apply the induction hypothesis to e_0.
- If $e = e_0 \parallel e_1$, then there exist $U_0 \in [\![e_0]\!]$ and $U_1 \in [\![e_1]\!]$ such that $V \cdot W \sqsubseteq U_0 \parallel U_1$. By Lemma 3.28, we find $V_0, V_1, W_0, W_1 \in \mathsf{SP}$ such that $V \sqsubseteq V_0 \parallel V_1$ and $W \sqsubseteq W_0 \parallel W_1$, as well as $V_i \cdot W_i \sqsubseteq U_i$ for all $i \in \{0, 1\}$. In that case, $V_i \cdot W_i \in [\![e_i]\!]{\downarrow}$ for all $i \in \{0, 1\}$, and thus by induction we find $\ell_i, r_i \in \mathcal{T}$ with $\ell_i \mathrel{\nabla_{e_i}} r_i$ such that $V_i \in [\![\ell_i]\!]{\downarrow}$ and $W_i \in [\![r_i]\!]{\downarrow}$. We choose $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$ to find that $V \in [\![\ell_0 \parallel r_0]\!]{\downarrow}$ and $W \in [\![\ell_1 \parallel r_1]\!]{\downarrow}$, as well as $\ell \mathrel{\nabla_e} r$.
- If $e = e_0^*$, then there exist $U_1, \dots, U_n \in [\![e_0]\!]{\downarrow}$ such that $V \cdot W = U_1 \cdots U_n$. Without loss of generality, we can assume that for $1 \leq i \leq n$ it holds that $U_i \neq 1$.

  We then proceed by induction on n, proving that we can find $\ell', r' \in \mathcal{T}$ such that $\ell' \mathrel{\nabla_{e_0}} r'$ with $V \in [\![e_0^* \cdot \ell']\!]{\downarrow}$ and $W \in [\![r' \cdot e_0^*]\!]{\downarrow}$; since in this case $e_0^* \cdot \ell' \mathrel{\nabla_e} r' \cdot e_0^*$ and $[\![\ell']\!]{\downarrow} \subseteq [\![e_0^* \cdot \ell']\!]{\downarrow}$, we can choose $\ell = e_0^* \cdot \ell'$ and $r = r' \cdot e_0^*$ to satisfy the claim. In the base, where $n = 0$, we have that $V \cdot W = 1$, thus $V = W = 1$; we can choose $\ell' = r' = 1$ to satisfy the claim.

  In the inductive step, we have $n > 0$, and assume that the claim holds for $n - 1$. By Lemma 3.A.1, we then have $X \in \mathsf{SP}$ such that one of the following two cases applies:

- If $V \cdot X = U_1 \cdots U_{n-1}$ and $X \cdot U_n = W$, then since $V \cdot X = U_1 \cdots U_{n-1} \in \llbracket e_0^* \rrbracket{\downarrow}$, we find by induction $\ell', r' \in \mathcal{T}$ such that $\ell' \nabla_{e_0} r'$ with $V \in \llbracket e_0^* \cdot \ell' \rrbracket{\downarrow}$ and $X \in \llbracket r' \cdot e_0^* \rrbracket{\downarrow}$. Since $W = X \cdot U_n \in \llbracket r' \cdot e_0^* \rrbracket{\downarrow} \cdot \llbracket e_0 \rrbracket{\downarrow} \subseteq \llbracket r' \cdot e_0^* \rrbracket{\downarrow}$, the claim follows.
- If $U_1 \cdots U_{n-1} \cdot X = V$ and $X \cdot W = U_n$, then since $X \cdot W = U_n \in \llbracket e_0 \rrbracket{\downarrow}$ we find by induction $\ell', r' \in \mathcal{T}$ such that $\ell' \nabla_{e_0} r'$ with $X \in \llbracket \ell' \rrbracket{\downarrow}$ and $W \in \llbracket r' \rrbracket{\downarrow}$. It then follows that $V = U_1 \cdots U_{n-1} \cdot X \in \llbracket e_0^* \rrbracket{\downarrow} \cdot \llbracket \ell' \rrbracket{\downarrow} = \llbracket e_0^* \cdot \ell' \rrbracket{\downarrow}$ and $W \in \llbracket r' \rrbracket{\downarrow} \subseteq \llbracket r' \cdot e_0^* \rrbracket{\downarrow}$. □

Lemma 5.28. *Let $e \in \mathcal{T}$. There exist $\ell_1, \dots, \ell_n \in \mathcal{T}$ and $r_1, \dots, r_n \in \mathcal{F}$ such that for $1 \leq i \leq n$ it holds that $\ell_i \nabla_e r_i$, and furthermore $e \equiv \ell_1 + \cdots + \ell_n$.*

Proof. We proceed by induction on e. In the base, there are two cases to consider.

- If $e = 0$, then we can choose $n = 0$ to satisfy the claim.
- If $e = \{1\} \cup \Sigma$, then we can choose $\ell_1 = r_1 = 1$ to satisfy the claim.

For the inductive step, there are five cases.

- If $e = e_0 + e_1$, then by induction there exist $\ell'_1, \dots, \ell'_{n'} \in \mathcal{T}$ and $\ell''_1, \dots, \ell'_{n''} \in \mathcal{T}$ as well as $r'_1, \dots, r'_{n'} \in \mathcal{F}$ as well as $r''_1, \dots, r''_{n''} \in \mathcal{F}$ such that for $1 \leq i \leq n'$ it holds that $\ell'_i \nabla_{e_0} r'_i$, and for $1 \leq i \leq n''$ it holds that $\ell''_i \nabla_{e_1} r''_i$, and moreover $e_0 \equiv \ell'_1 + \cdots + \ell'_{n'}$ as well as $e_1 \equiv \ell''_1 + \cdots + \ell''_{n''}$. We can then choose $n = n' + n''$ where for $1 \leq i \leq n'$ we have $\ell_i = \ell'_i$ and $r_i = r'_i$, as well as for $n' < i \leq n$ we have $\ell_i = \ell''_{n'-i}$ and $r_i = r''_{n'-i}$ to satisfy the claim. After all, for $1 \leq i \leq n$ we have $\ell_i \nabla_e r_i$, and furthermore

$$e = e_0 + e_1 \equiv \ell'_1 + \cdots + \ell_{n'} + \ell''_1 + \cdots + \ell''_{n''} = \ell_1 + \cdots + \ell_n$$

- If $e = e_0 \cdot e_1$, then by induction there exist $\ell'_1, \dots, \ell'_{n'} \in \mathcal{T}$ as well as $r'_1, \dots, r'_{n'} \in \mathcal{F}$ such that for $1 \leq i \leq n'$ it holds that $\ell'_i \nabla_{e_1} r'_i$. We can then choose $n = n'$ and for $1 \leq i \leq n$ set $\ell_i = e_0 \cdot \ell'_i$ as well as $r_i = r'_i$ to find that for $1 \leq i \leq n$ we have $\ell_i \nabla_e r_i$, and furthermore

$$e = e_0 \cdot e_1 \equiv (\ell'_1 + \cdots + \ell'_{n'}) \equiv e_0 \cdot \ell'_1 + \cdots + e_0 \cdot \ell'_{n'} = \ell_1 + \cdots + \ell_n$$

- If $e = e_0 \parallel e_1$, then by induction there exist $\ell'_1, \dots, \ell'_{n'} \in \mathcal{T}$ and $\ell''_1, \dots, \ell'_{n''} \in \mathcal{T}$ as well as $r'_1, \dots, r'_{n'} \in \mathcal{F}$ as well as $r''_1, \dots, r''_{n''} \in \mathcal{F}$ such that for $1 \leq i \leq n'$ it holds that $\ell'_i \nabla_{e_0} r'_i$, and for $1 \leq i \leq n''$ it holds that $\ell''_i \nabla_{e_1} r''_i$, and moreover $e_0 \equiv \ell'_1 + \cdots + \ell'_{n'}$ as well as $e_1 \equiv \ell''_1 + \cdots + \ell''_{n''}$. We can then choose $n = n' \cdot n''$ as well as for $1 \leq i \leq n'$ and $1 \leq j \leq n''$

that $\ell_{i+(j-1)\cdot n'} = \ell'_i \parallel \ell''_j$ and $r_{i+(j-1)\cdot n'} = r'_i \parallel r''_j \in \mathcal{F}$, to find that for $1 \leq i \leq n'$ and $1 \leq j \leq n''$ we have that $\ell_{i+(j-1)\cdot n'} = \ell'_i \parallel \ell''_j \nabla_e r'_i \parallel r''_j = r_{i+(j-1)\cdot n'}$, and furthermore

$$e = e_0 \parallel e_1 \equiv (\ell'_1 + \cdots + \ell''_{n'}) \parallel (\ell''_1 + \cdots + \ell''_{n''}) \equiv \ell'_1 \parallel \ell''_1 + \cdots + \ell'_{n'} \parallel \ell'_{n''} \equiv \ell_1 + \cdots + \ell_n$$

- If $e = e_0^*$, then by induction there exist $\ell'_1, \ldots, \ell'_{n'} \in \mathcal{T}$ as well as $r'_1, \ldots, r'_{n'} \in \mathcal{F}$ such that for $1 \leq i \leq n'$ it holds that $\ell'_i \nabla_{e_0} r'_i$. We can then choose $n = n' + 1$ and for $1 \leq i \leq n$ set $\ell_i = e_0^* \cdot \ell'_i$ and $r_i = r'_i \cdot e_0^* \in \mathcal{F}$, as well as $\ell_n = r_n = 1 \in \mathcal{F}$, to find that for $1 \leq i \leq n$ we have $\ell_i = e_0^* \cdot \ell'_i \nabla_e r'_i \cdot e_0^*$, and furthermore that

$$e \equiv e_0 \cdot e_0^* + 1 \equiv (\ell'_1 + \cdots + \ell_{n'}) \cdot e_0^* + 1 \equiv \ell'_1 \cdot e_0^* + \cdots + \ell'_{n'} \cdot e_0^* + 1 \equiv \ell_1 + \cdots + \ell_{n'} + \ell_n \quad \square$$

Lemma 5.29. *Let $e \in \mathcal{T}$. The following hold.*

(i) There are finitely many $\ell, r \in \mathcal{T}$ such that $\ell \nabla_e r$.

(ii) If $\ell \nabla_e r$, then $\ell \cdot r \leqq^{\mathsf{exch}} e$.

(iii) If $\ell \nabla_e r$, then $\mathsf{d}_\parallel(\ell), \mathsf{d}_\parallel(r) \leq \mathsf{d}_\parallel(e)$.

Proof. We treat the claims in the order given.

- The proof proceeds by induction on e, deriving a constraint on ℓ and r that is satisfied by only finitely many sr-expressions. In the base, we can disregard the case where $e = 0$, for no rule applies here. This leaves us two cases to consider.

  - If $e = 1$, then $\ell = r = 1$.
  - If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $\langle \ell, r \rangle \in \{\langle 1, \mathtt{a}\rangle, \langle \mathtt{a}, 1\rangle\}$.

  In the inductive step, suppose that $\ell, r \in \mathcal{T}$ are such that $\ell \nabla_e r$. There are four cases.

  - If $e = e_0 + e_1$, then $\ell \nabla_{e_i} r$ for some $i \in \{0, 1\}$.
  - If $e = e_0 \cdot e_1$, then either $\ell = e_0 \cdot \ell'$ and $\ell' \nabla_{e_1} r$, or $r = r' \cdot e_1$ and $\ell \nabla_{e_0} r'$.
  - If $e = e_0 \parallel e_1$, then $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$, such that for $i \in \{0, 1\}$ we have $\ell_i \nabla_{e_i} r_i$.
  - If $e = e_0^*$, then either $\ell = r = 1$, or $\ell = e \cdot \ell'$ and $r = r' \cdot e$ such that $\ell \nabla_{e_0} r$

  In all cases, there are finitely many $\ell, r \in \mathcal{T}$ that satisfy the restrictions put on them, by induction. For instance, in the first case we know by induction that there are finitely many $\ell, r \in \mathcal{T}$ such that $\ell \nabla_{e_0} r$ or $\ell \nabla_{e_1} r$. This completes the proof of this claim.

- The proof proceeds by induction on the construction of ∇. In the base, there are three cases.

- If $e = \ell = r = 1$, then $\ell \cdot r \equiv e$, and so the claim holds immediately.
- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and either $\ell = 1$ and $r = \mathtt{a}$, or $\ell = \mathtt{a}$ and $r = 1$, then $\ell \cdot r \equiv e$.
- If $e = e_0^*$ and $\ell = r = 1$, then $\ell \cdot r \equiv 1 \leqq e_0^* = e$, and so the claim holds.

For the inductive step, there are four cases to consider.

- If $e = e_0 + e_1$ and $\ell \nabla_{e_i} r$ for some $i \in \{0, 1\}$, then $\ell \cdot r \leqq^{\mathsf{exch}} e_i \leqq e$ by induction.
- If $e = e_0 \cdot e_1$ and $r = r' \cdot e_1$ with $\ell \nabla_{e_0} r'$, then by induction we find that $\ell \cdot r' \leqq^{\mathsf{exch}} e_0$. It then follows that $\ell \cdot r = \ell \cdot r' \cdot e_1 \leqq e_0 \cdot e_1 = e$. The case where $e = e_0 \cdot e_1$ and $\ell = e_0 \cdot \ell'$ with $\ell' \nabla_{e_1} r$ can be treated similarly.
- If $e = e_0 \parallel e_1$ and $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$ such that $\ell_i \nabla_{e_i} r_i$ for all $i \in \{0, 1\}$, then by induction we have that $\ell_i \cdot r_i \leqq^{\mathsf{exch}} e_i$ for all $i \in \{0, 1\}$. We then find that

$$\ell \cdot r = (\ell_0 \parallel \ell_1) \cdot (r_0 \parallel r_1) \leqq^{\mathsf{exch}} (\ell_0 \cdot r_0) \parallel (\ell_1 \cdot r_1) \leqq e_0 \parallel e_1 = e$$

- If $e = e_0^*$ and $\ell = e_0^* \cdot \ell'$ and $r = r' \cdot e_0^*$ such that $\ell' \nabla_{e_0} r'$, then by induction we have that $\ell' \cdot r' \leqq^{\mathsf{exch}} e_0$. This allows us to derive that $\ell \cdot r = e_0^* \cdot \ell' \cdot r' \cdot e_0^* \leqq e_0^* \cdot e_0 \cdot e_0^* \leqq e_0^* = e$.

• The proof proceeds by induction on the construction of ∇. In the base, there are three cases.

- If $e = \ell = r = 1$, then the claim holds immediately.
- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and either $\ell = 1$ and $r = \mathtt{a}$ or $\ell = \mathtt{a}$ and $r = 1$, then $\mathsf{d}_{\parallel}(\ell) = \mathsf{d}_{\parallel}(r) = 0 = \mathsf{d}_{\parallel}(e)$, and so the claim holds again.
- If $e = e_0^*$ and $\ell = r = 1$, then $\mathsf{d}_{\parallel}(\ell) = \mathsf{d}_{\parallel}(r) = 0 \leq \mathsf{d}_{\parallel}(e_0^*)$ immediately.

For the inductive step, there are four cases to consider.

- If $e = e_0 + e_1$ and $\ell \nabla_{e_i} r$ for some $i \in \{0, 1\}$, then $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r) \leq \mathsf{d}_{\parallel}(e_i)$ by induction. Since $\mathsf{d}_{\parallel}(e_i) \leq \mathsf{d}_{\parallel}(e)$, the claim then follows.
- If $e = e_0 \cdot e_1$ and $r = r' \cdot e_1$ with $\ell \nabla_{e_0} r'$, then by induction we find that $\mathsf{d}_{\parallel}(\ell), \mathsf{d}_{\parallel}(r') \leq \mathsf{d}_{\parallel}(e_0)$. Since $\mathsf{d}_{\parallel}(e_0) \leq \mathsf{d}_{\parallel}(e)$, we then immediately know that $\mathsf{d}_{\parallel}(\ell) \leq \mathsf{d}_{\parallel}(e)$. Furthermore, since $\mathsf{d}_{\parallel}(r) = \max(\mathsf{d}_{\parallel}(r'), \mathsf{d}_{\parallel}(e_1)) \leq \max(\mathsf{d}_{\parallel}(e_0), \mathsf{d}_{\parallel}(e_1)) = \mathsf{d}_{\parallel}(e)$, other inequality also holds. The case where $e = e_0 \cdot e_1$ and $\ell = e_0 \cdot \ell'$ with $\ell' \nabla_{e_1} r$ can be treated similarly.
- If $e = e_0 \parallel e_1$, then $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$ such that $\ell_i \nabla_{e_i} r_i$ for $i \in \{0, 1\}$. By induction, we then have that $\mathsf{d}_{\parallel}(\ell_i), \mathsf{d}_{\parallel}(r_i) \leq \mathsf{d}_{\parallel}(e_i)$ for $i \in \{0, 1\}$. We can then derive that

$$\mathsf{d}_{\parallel}(\ell) = \max(\mathsf{d}_{\parallel}(\ell_0), \mathsf{d}_{\parallel}(\ell_1)) + 1 \leq \max(\mathsf{d}_{\parallel}(e_0), \mathsf{d}_{\parallel}(e_1)) + 1 = \mathsf{d}_{\parallel}(e)$$

and similarly show that $\mathsf{d}_{\|}(r) \leq \mathsf{d}_{\|}(e)$.

– If $e = e_0^*$ with $\ell = e_0^* \cdot \ell'$ and $r = r' \cdot e_0^*$ such that $\ell' \nabla_{e_0} r'$, then by induction we know that $\mathsf{d}_{\|}(\ell'), \mathsf{d}_{\|}(r') \leq \mathsf{d}_{\|}(e_0)$. We can then derive that

$$\mathsf{d}_{\|}(\ell) = \max(\mathsf{d}_{\|}(e_0^*), \mathsf{d}_{\|}(\ell')) \leq \max(\mathsf{d}_{\|}(e_0^*), \mathsf{d}_{\|}(e_0)) = \mathsf{d}_{\|}(e_0^*)$$

and similarly for $\mathsf{d}_{\|}(r)$. □

Lemma 5.32. *Let $e \in \mathcal{T}$. $R(e)$ is finite. Furthermore, if $e' \in R(e)$, then $\mathsf{d}_{\|}(e') \leq \mathsf{d}_{\|}(e)$.*

Proof. For finiteness, it suffices to find for every $e \in \mathcal{T}$ a finite set $S(e)$ that contains e and satisfies the rule that defines $R(e)$ — after all, since $R(e)$ is the smallest such set, it follows that $R(e) \subseteq S(e)$, and hence $R(e)$ must be finite. To this end, we define for $e \in \mathcal{T}$ the set $S(e)$ inductively, as follows

$$S(0) = \{0\} \qquad S(e_0 + e_1) = \{e_0 + e_1\} \cup S(e_0) \cup S(e_1)$$
$$S(1) = \{1\} \qquad S(e_0 \cdot e_1) = \{e_0' \cdot e_1 \;:\; e_0' \in S(e_0')\} \cup S(e_1)$$
$$S(\mathsf{a}) = \{\mathsf{a}, 1\} \qquad S(e_0 \parallel e_1) = \{e_0' \parallel e_1' \;:\; e_0' \in S(e_0), e_1' \in S(e_1)\}$$
$$S(e_0^*) = \{1, e_0^*\} \cup \{e_0' \cdot e_0^* \;:\; e_0' \in S(e_0)\}$$

It should be clear that $S(e)$ is finite for every $e \in \mathcal{T}$. To show that $S(e)$ is a valid choice for our objective, we proceed by induction on e. In the base, there are three cases to consider, and in each case the claim holds trivially. For the inductive step, there are four cases to consider.

- If $e = e_0 + e_1$, then note that $e_0 + e_1 \in S(e)$ by definition. Furthermore, if $e' \in S(e)$ and $\ell \nabla_{e'} r$, then there are two cases to consider. On the one hand, if $e' = e_0 + e_1$, then $\ell \nabla_{e_0} r$ or $\ell \nabla_{e_1} r$, which means that $r \in S(e_0)$ or $r \in S(e_1)$; in either case $r \in S(e)$. On the other hand, if $e' \in S(e_0)$ or $e' \in S(e_1)$, then $r \in S(e_0)$ or $r \in S(e_1)$ by induction.

- If $e = e_0 \cdot e_1$, then note that since $e_0 \in S(e_0)$, we also have that $e \in S(e)$. Now, if $e' \in S(e)$, then $e' = e_0' \cdot e_1$ for some $e_0' \in S(e_0)$. Suppose that $\ell \nabla_{e'} r$; then there are two cases. On the one hand, if $r = r' \cdot e_1$ such that $\ell \nabla_{e_0'} r'$, then by induction we know that $r' \in S(e_0)$, and hence $r = r' \cdot e_1 \in S(e)$. On the other hand, if $\ell = e_0' \cdot \ell'$ and $\ell' \nabla_{e_1} r$, then $r \in S(e_1)$ by induction, and hence $r \in S(e)$ again.

- If $e = e_0 \parallel e_1$, then note that since $e_0 \in S(e_0)$ and $e_1 \in S(e_1)$, we also have that $e \in S(e)$. Now, if $e' \in S(e)$, then $e' = e_0' \parallel e_1'$ with $e_0' \in S(e_0)$ and $e_1' \in S(e_1)$. Suppose that $\ell \nabla_{e'} r'$; then $\ell = \ell_0 \parallel \ell_1$ and $r = r_0 \parallel r_1$, and $\ell_i \nabla_{e_i} r_i$ for $i \in \{0, 1\}$. By induction, $r_0 \in S(e_0)$ and $r_1 \in S(e_1)$, which means that $r = r_0 \parallel r_1 \in S(e)$.

- If $e = e_0^*$, then note that $e_0^* \in S(e)$ by construction. Now, if $e' \in S(e)$ and $\ell \nabla_e r$, then there are three subcases to consider.
  - If $e' = 1$, then $\ell = r = 1$, in which case $r \in S(e)$ immediately.
  - If $e' = e_0^*$, then $\ell = r = 1$, or $\ell = e_0^* \cdot \ell'$ and $r = r' \cdot e_0^*$ with $\ell' \nabla_{e_0} r'$. In the former case, $r \in S(e)$. In the latter case, $r' \in S(e_0)$ by induction, and hence $r = r' \cdot e_0^* \in S(e)$.
  - If $e' = e_0' \cdot e_0^*$ with $e_0' \in S(e_0)$, then either $r = r' \cdot e_0^*$ with $\ell \nabla_{e_0'} r'$, or $\ell = e_0^* \cdot \ell'$ and $\ell' \nabla_{e_0^*} r$. In the former case, $r' \in S(e_0)$, meaning that $r = r' \cdot e_0^* \in S(e)$. In the latter case, $r \in S(e)$ by an argument similar to the previous case.

For the second claim, we show that if $e' \in S(e)$, then $\mathsf{d}_{\parallel}(e') \leq \mathsf{d}_{\parallel}(e)$, by induction on e. In the base, where $e \in \{0, 1\} \cup \Sigma$, the claim holds, for $\mathsf{d}_{\parallel}(e') = 0$. For the inductive step, there are four cases.

- If $e = e_0 + e_1$, then either $e' = e_0 + e_1$, in which case the claim holds immediately, or $e' \in S(e_0)$ or $e' \in S(e_1)$. If $e' \in S(e_0)$, then $\mathsf{d}_{\parallel}(e') \leq \mathsf{d}_{\parallel}(e_0)$ by induction. Since $\mathsf{d}_{\parallel}(e_0) \leq \mathsf{d}_{\parallel}(e)$, the claim follows; the case where $e' \in S(e_1)$ can be argued similarly.
- If $e = e_0 \cdot e_1$, then either $e' = e_0' \cdot e_1$ for $e_0' \cdot S(e_0)$, or $e' \in S(e_1)$. In the former case, $\mathsf{d}_{\parallel}(e_0') \leq \mathsf{d}_{\parallel}(e_0)$ by induction, and hence $\mathsf{d}_{\parallel}(e') = \max(\mathsf{d}_{\parallel}(e_0'), \mathsf{d}_{\parallel}(e_1)) \leq \max(\mathsf{d}_{\parallel}(e_0), \mathsf{d}_{\parallel}(e_1)) = \mathsf{d}_{\parallel}(e)$. In the latter case, $\mathsf{d}_{\parallel}(e') \leq \mathsf{d}_{\parallel}(e_1)$ by induction; since $\mathsf{d}_{\parallel}(e_1) \leq \mathsf{d}_{\parallel}(e)$, the claim follows.
- If $e = e_0 \parallel e_1$, then $e' = e_0' \parallel e_1'$ for $e_0' \in S(e_0)$ and $e_1' \in S(e_1)$. By induction, we then know that $\mathsf{d}_{\parallel}(e_0') \leq \mathsf{d}_{\parallel}(e_0)$ and $\mathsf{d}_{\parallel}(e_1') \leq \mathsf{d}_{\parallel}(e_1')$, whence $\mathsf{d}_{\parallel}(e') = \max(\mathsf{d}_{\parallel}(e_0'), \mathsf{d}_{\parallel}(e_1')) + 1 \leq \max(\mathsf{d}_{\parallel}(e_0), \mathsf{d}_{\parallel}(e_1)) + 1 = \mathsf{d}_{\parallel}(e)$.
- If $e = e_0^*$, then either $e' = 1$, in which case $\mathsf{d}_{\parallel}(e') = 0 \leq \mathsf{d}_{\parallel}(e)$ immediately, or $e' = e$, in which case $\mathsf{d}_{\parallel}(e') \leq \mathsf{d}_{\parallel}(e)$, or $e' = e_0' \cdot e_0^*$ for some $e_0' \in S(e_0)$. In the latter case, $\mathsf{d}_{\parallel}(e_0') \leq \mathsf{d}_{\parallel}(e_0)$ by induction. We then derive $\mathsf{d}_{\parallel}(e') = \max(\mathsf{d}_{\parallel}(e_0'), \mathsf{d}_{\parallel}(e_0^*)) \leq \max(\mathsf{d}_{\parallel}(e_0), \mathsf{d}_{\parallel}(e_0^*)) = \mathsf{d}_{\parallel}(e)$. □

5.B Proofs to factorise the exchange law

Lemma 5.47. *Let $C \in \mathsf{PC}$; the following hold:*

(i) $C \in \mathsf{PC}^{\mathsf{sp}}$ if and only if C is series-parallel, and

(ii) if $C = [\mathbf{c}] \in \mathsf{PC}^{\mathsf{sp}}$ and $U = [\mathbf{u}] \in \mathsf{Pom}$, then $C[U] = [\mathbf{c}[\mathbf{u}]]$.

Proof. We treat the claims in the order given.

(i) The implication from left to right holds by construction of $\mathsf{PC}^{\mathsf{sp}}$. For the other implication, we proceed by induction on the construction of C as a series-parallel pomset. In the base, we can disregard the case where $C = 1$, for then C would not have any node labelled with $\square$, meaning that $C \notin \mathsf{PC}$. We thus know that C is primitive; it must then be the case that $C = \square$ (otherwise, C would again not be a valid pomset context), which means that $C \in \mathsf{PC}^{\mathsf{sp}}$.

For the inductive step, there are two cases to consider. If $C = V \cdot W$, then exactly one of V and W contains the node labelled by $\square$, and the other one does not have any nodes labelled by $\square$. Thus, if V has exactly one node labelled by $\square$, then $V \in \mathsf{PC}$. Since furthermore both V and W are series-parallel, it follows that $V \in \mathsf{PC}^{\mathsf{sp}}$ by induction, and hence $C = V \cdot W \in \mathsf{PC}^{\mathsf{sp}}$. The case where V has exactly one node labelled by $\square$ is similar, as is the case where $C = V \parallel W$.

(ii) We proceed by induction on the construction of C as a member of $\mathsf{PC}^{\mathsf{sp}}$. In the base, $C = \square$; in this case, the definition of $\mathbf{c}[\mathbf{u}]$ degenerates into $\mathbf{u}$. We check this as follows:

- For the carrier, we have $S_{\mathbf{c}[\mathbf{u}]} = S_{\mathbf{u}} \cup S_{\mathbf{c}} \setminus \{s_\square\} = S_{\mathbf{u}} \cup \{s_\square\} \setminus \{s_\square\} = S_{\mathbf{u}} \cup \emptyset = \S_{\mathbf{u}}$.
- For the order, we claim that $\leq_{\mathbf{u}}$ is the same as $\leq_{\mathbf{c}[\mathbf{u}]}$. The inclusion from left to right follows by definition of $\leq_{\mathbf{c}[\mathbf{u}]}$. For the other inclusion, we note that only the first rule in the construction of $\leq_{\mathbf{c}[\mathbf{u}]}$ has any instances; after all, if $s \leq_{\mathbf{c}[\mathbf{u}]} s'$, then $s, s' \in S_{\mathbf{c}[\mathbf{u}]}$, which means in particular that $s, s' \notin S_{\mathbf{c}} = \{s_\square\}$.
- For the labelling, we note that if $s \in S_{\mathbf{c}[\mathbf{u}]} = S_{\mathbf{u}}$, then $\lambda_{\mathbf{c}[\mathbf{u}]}(s) = \lambda_{\mathbf{u}}(s)$ by definition.

For the inductive step, we consider the case where $C = C' \cdot V$ for $C' \in \mathsf{PC}^{\mathsf{sp}}$ and $V \in \mathsf{SP}$; the other cases are similar. We write $C' = [\mathbf{c}']$ and $V = [\mathbf{v}]$, noting $\mathbf{c} = \mathbf{c}' \cdot \mathbf{v}$. It suffices to show that $\mathbf{c}[\mathbf{u}] = \mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}$, which we do by comparing their components as labelled posets.

- For the carriers, we derive that they coincide as follows:

$$S_{\mathbf{c}[\mathbf{u}]} = S_{\mathbf{c}} \cup S_{\mathbf{u}} - \{s_\square\} = S_{\mathbf{c}'} \cup S_{\mathbf{v}} \cup S_{\mathbf{u}} - \{s_\square\} = S_{\mathbf{c}'[\mathbf{u}]} \cup S_{\mathbf{v}} = S_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}}$$

- For the ordering, first suppose that $s, s' \in S_{\mathbf{c}[\mathbf{u}]}$ with $s \leq_{\mathbf{c}[\mathbf{u}]} s'$.
  - If $s, s' \in S_{\mathbf{c}} - \{s_\square\}$, then $s \leq_{\mathbf{c}} s'$; this gives us three subcases to consider.
    - ∗ If $s, s' \in S_{\mathbf{c}'} - \{s_\square\}$, then $s \leq_{\mathbf{c}'} s'$, meaning $s \leq_{\mathbf{c}'[\mathbf{u}]} s'$; thus, $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$.
    - ∗ If $s, s' \in S_{\mathbf{v}}$, then $s \leq_{\mathbf{v}} s'$, meaning that $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$.
    - ∗ If $s \in S_{\mathbf{c}'} - \{s_\square\}$ and $s' \in S_{\mathbf{v}}$, then $s \in S_{\mathbf{c}'[\mathbf{u}]}$, meaning $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$.
  - If $s, s' \in S_{\mathbf{u}}$, then $s \leq_{\mathbf{u}} s'$. This tells us that $s \leq_{\mathbf{c}'[\mathbf{u}]} s'$, meaning $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$.
  - If $s \in S_{\mathbf{u}}$ and $s' \in S_{\mathbf{c}} - \{s_\square\}$ with $s_\square \leq_{\mathbf{c}} s'$, then there are two subcases:

    * If $s' \in S_{\mathbf{c}'} - \{s_\square\}$, then $s_\square \leq_{\mathbf{c}'} s'$, meaning $s \leq_{\mathbf{c}'[\mathbf{u}]} s'$; thus, $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$.
    * If $s' \in S_{\mathbf{v}}$, then since $s \in S_{\mathbf{c}'[\mathbf{u}]}$, we have $s \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}} s'$ immediately.
  - If $s \in S_{\mathbf{c}}$ and $s' \in S_{\mathbf{u}}$ with $s \leq_{\mathbf{c}} s_\square$, an argument similar to the above applies.

  This shows that $\leq_{\mathbf{c}[\mathbf{u}]} \subseteq \leq_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}}$; the other inclusion can be shown similarly.

- To see that their labellings are the same, suppose that $s \in S_{\mathbf{c}[\mathbf{u}]}$; there are three cases.
  - If $s \in S_{\mathbf{c}'} - \{s_\square\}$, then in particular $s \in S_{\mathbf{c}} - \{s_\square\}$, meaning

$$\lambda_{\mathbf{c}[\mathbf{u}]}(s) = \lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{c}'}(s) = \lambda_{\mathbf{c}'[\mathbf{u}]}(s) = \lambda_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}}(s)$$

  - If $s \in S_{\mathbf{v}}$, then $s \in S_{\mathbf{c}} - \{s_\square\}$, whence $\lambda_{\mathbf{c}[\mathbf{u}]}(s) = \lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{v}}(s) = \lambda_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}}(s)$.
  - If $s \in S_{\mathbf{u}}$, then we derive $\lambda_{\mathbf{c}[\mathbf{u}]}(s) = \lambda_{\mathbf{u}}(s) = \lambda_{\mathbf{c}'[\mathbf{u}]}(s) = \lambda_{\mathbf{c}'[\mathbf{u}] \cdot \mathbf{v}}(s)$. □

To prove Lemma 5.48, we need the following technical lemma.

Lemma 5.B.1. *Let $C \in \mathsf{PC}$ and $U \in \mathsf{Pom}$ and $\mathtt{a} \in \Sigma$. Now $C[\mathtt{a}] = U$ if and only if $C = [\mathbf{c}]$ and $U = [\mathbf{u}]$ such that all of the following hold:*

(i) $S_{\mathbf{c}} = S_{\mathbf{u}}$ *as well as* $\leq_{\mathbf{c}} = \leq_{\mathbf{u}}$, *and*

(ii) $\lambda_{\mathbf{c}}(s_\square) = \square$ *and* $\lambda_{\mathbf{u}}(s_\square) = \mathtt{a}$, *and*

(iii) $\lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s)$ *for all* $s \in S_{\mathbf{c}} - \{s_\square\}$.

Proof. Let $C = [\mathbf{c}]$ with $s_\square \in S_{\mathbf{c}}$ the unique node of $\mathbf{c}$ such that $\lambda_{\mathbf{c}}(s_\square) = \square$. Also, let $\mathtt{a} = [\mathbf{a}]$, where we assume without loss of generality that $S_{\mathbf{a}} = \{s_{\mathtt{a}}\}$ (where $s_{\mathtt{a}} \notin S_{\mathbf{c}}$), and we have $\lambda_{\mathbf{a}}(s_{\mathtt{a}}) = \mathtt{a}$.

For the direction from left to right, we choose $S_{\mathbf{u}} = S_{\mathbf{c}}$ and $\leq_{\mathbf{u}} = \leq_{\mathbf{c}}$, and we set $\lambda_{\mathbf{u}}(s_\square) = \mathtt{a}$, while $\lambda_{\mathbf{u}}(s) = \lambda_{\mathbf{c}}(s)$ for all $s \in S_{\mathbf{c}} - \{s_\square\}$. It should be clear that this choice of $\mathbf{u}$ and $\mathbf{c}$ satisfies the three conditions above; it remains to prove that $[\mathbf{u}] = U$, for which it suffices to show that that $\mathbf{u}$ is isomorphic to $\mathbf{c}[\mathbf{a}]$, since $C[\mathtt{a}] = U$. To see this, first note that

$$S_{\mathbf{c}[\mathbf{a}]} = S_{\mathbf{c}} \cup S_{\mathbf{a}} - \{s_\square\} = S_{\mathbf{c}} \cup \{s_{\mathtt{a}}\} - \{s_\square\}$$

We choose $h : S_{\mathbf{c}[\mathbf{a}]} \to S_{\mathbf{u}}$ by setting $h(s_{\mathtt{a}}) = s_\square$ and $h(s) = s$ when $s \neq s_{\mathtt{a}}$; clearly, h is a bijection between $S_{\mathbf{c}[\mathbf{a}]}$ and $S_{\mathbf{u}}$. To see that h preserves labels, first note

$$\lambda_{\mathbf{u}}(h(s_{\mathtt{a}})) = \lambda_{\mathbf{u}}(s_\square) = \mathtt{a} = \lambda_{\mathbf{a}}(s_{\mathtt{a}}) = \lambda_{\mathbf{c}[\mathbf{a}]}(s_{\mathtt{a}})$$

Second, when $s \neq s_{\mathtt{a}}$ we have that $s \in S_{\mathbf{c}}$, and hence $\lambda_{\mathbf{u}}(h(s)) = \lambda_{\mathbf{u}}(s) = \lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{c}[\mathbf{a}]}(s)$.

To see that h preserves order, suppose that $s, s' \in S_{\mathbf{c}[\mathbf{a}]}$ such that $s \leq_{\mathbf{c}[\mathbf{a}]} s'$; there are four cases.

- If $s \leq_{\mathbf{c}} s'$, then $s, s' \neq s_{\mathsf{a}}$, and hence $h(s) = s \leq_{\mathbf{u}} s' = h(s')$ by definition.
- If $s \leq_{\mathbf{a}} s'$, then $s = s' = s_{\mathsf{a}}$ since $S_{\mathbf{a}}$ is a singleton; hence $h(s_{\mathsf{a}}) \leq_{\mathbf{u}} h(s_{\mathsf{a}})$.
- If $s \leq_{\mathbf{c}} s_{\square}$ and $s' \in S_{\mathbf{a}}$, then $s \in S_{\mathbf{c}}$ and $s' = s_{\mathsf{a}}$; hence $h(s) = s \leq_{\mathbf{u}} s_{\square} = h(s')$.
- If $s_{\square} \leq_{\mathbf{c}} s'$ and $s \in S_{\mathbf{a}}$, then $s' \in S_{\mathbf{c}}$ and $s = s_{\mathsf{a}}$; hence $h(s) = s_{\square} \leq_{\mathbf{u}} s' = h(s')$.

A similar argument shows that h reflects ordering; hence, h is a pomset isomorphism, and $[\mathbf{c}[\mathbf{a}]] = [\mathbf{u}]$.

For the converse direction, suppose that $U = [\mathbf{u}]$ such that the three conditions above are satisfied. It remains to show that $C[\mathsf{a}] = U$ — in other words, that $\mathbf{c}[\mathbf{a}]$ is isomorphic to $\mathbf{u}$. As isomorphism, we choose the identity function, which is already a bijection by the first property; it also preserves and reflects ordering (by the first property), and preserves labels (by the second and last properties). Hence, $\mathbf{c}[\mathbf{a}]$ is isomorphic to $\mathbf{u}$, and therefore $C[\mathsf{a}] = U$. $\square$

Lemma 5.48. *Let $C \in \mathsf{PC}$, $V \in \mathsf{Pom}$ and $\mathsf{a} \in \Sigma$. The following hold:*

(i) If $C[\mathsf{a}] \sqsubseteq V$, then we can construct $C' \in \mathsf{PC}$ s.t. $C \sqsubseteq C'$ and $C'[\mathsf{a}] = V$.

(ii) If $V \sqsubseteq C[\mathsf{a}]$, then we can construct $C' \in \mathsf{PC}$ s.t. $C' \sqsubseteq C$ and $C'[\mathsf{a}] = V$.

Moreover, if $V \in \mathsf{SP}$, then the C' constructed in each of the above is series-parallel.

Proof. We prove the first claim; the second claim can be proved similarly. Let $U = C[\mathsf{a}]$. By Lemma 5.B.1, we have $C = [\mathbf{c}]$ and $U = [\mathbf{u}]$ with $S_{\mathbf{u}} = S_{\mathbf{c}}$ and $\leq_{\mathbf{u}} = \leq_{\mathbf{c}}$, with $\lambda_{\mathbf{c}}(s_{\square}) = \square$ and $\lambda_{\mathbf{u}}(s_{\square}) = \mathsf{a}$, and that $\lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s)$ for all $s \in S_{\mathbf{c}} - \{s_{\square}\}$. Without loss of generality, we can assume that $V = [\mathbf{v}]$ with $S_{\mathbf{v}} = S_{\mathbf{u}}$ and $\lambda_{\mathbf{v}} = \lambda_{\mathbf{u}}$ and $\leq_{\mathbf{u}} \subseteq \leq_{\mathbf{v}}$. We choose $S_{\mathbf{c}'} = S_{\mathbf{c}}$ and $\leq_{\mathbf{c}'} = \leq_{\mathbf{v}}$ and $\lambda_{\mathbf{c}'} = \lambda_{\mathbf{c}}$ to obtain $C' = [\mathbf{c}']$. First, note that $C' \sqsubseteq C$ by construction. Also, observe that $S_{\mathbf{c}'} = S_{\mathbf{c}} = S_{\mathbf{u}} = S_{\mathbf{v}}$; furthermore, $\lambda_{\mathbf{c}'}(s_{\square}) = \lambda_{\mathbf{c}}(s_{\square}) = *$, while $\lambda_{\mathbf{v}}(s_{\square}) = \lambda_{\mathbf{u}}(s_{\square}) = \mathsf{a}$, and for $s \in S_{\mathbf{c}} - \{s_{\square}\}$ we have $\lambda_{\mathbf{c}'}(s) = \lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s) = \lambda_{\mathbf{v}}(s)$. By Lemma 5.B.1, we conclude $C'[\mathsf{a}] = V$.

Finally, if $V \in \mathsf{SP}$, then C' must also be N-free (and hence series-parallel), since any N in C' must also occur in V (by construction of C'), and V is N-free because it is series-parallel. $\square$

To prove Lemma 5.49, we need the following auxiliary lemma.

Lemma 5.B.2. *Let $C \in \mathsf{PC}$ and $U \in \mathsf{Pom}$. Now $C[1] = U$ if and only if $C = [\mathbf{c}]$ and $U = [\mathbf{u}]$ s.t.:*

(i) $S_{\mathbf{c}} = S_{\mathbf{u}} \cup \{s_{\square}\}$ with $s_{\square} \notin S_{\mathbf{u}}$, and

(ii) for all $s, s' \in S_{\mathbf{u}}$ it holds that $s \leq_{\mathbf{u}} s'$ if and only if $s \leq_{\mathbf{c}} s'$, and

(iii) $\lambda_{\mathbf{c}}(s_{\square}) = \square$ and $\lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s)$ for all $s \in S_{\mathbf{u}}$.

Proof. Let $C = [\mathbf{c}]$, where $s_\Box \in S_\mathbf{c}$ is the unique $\Box$-labelled note of $\mathbf{c}$, and write $\mathbf{1}$ for the (unique) empty labelled partial order. For the direction from left to right, we know that $U = C[1] = [\mathbf{c}[\mathbf{1}]]$. It suffices to prove that $\mathbf{c}[\mathbf{1}]$ satisfies exactly the properties of $\mathbf{u}$ listed above. First of all, note that $S_{\mathbf{c}[\mathbf{1}]} = S_\mathbf{c} \cup S_\mathbf{1} - \{s_\Box\} = S_\mathbf{c} \setminus \{s_\Box\}$ by definition, hence $s_\Box \notin S_{\mathbf{c}[\mathbf{1}]}$. Furthermore, $S_{\mathbf{c}[\mathbf{1}]} \cup \{s_\Box\} = S_\mathbf{c}$. Next, suppose that $s, s' \in S_{\mathbf{c}[\mathbf{1}]}$ with $s \leq_{\mathbf{c}[\mathbf{1}]} s'$. We can discount the possibility that $s \in S_\mathbf{1}$ or $s' \in S_\mathbf{1}$, which leaves us to conclude that $s \leq_\mathbf{c} s'$; the converse holds by definition of $\leq_{\mathbf{c}[\mathbf{1}]}$. Lastly, note that $\lambda_\mathbf{c}(s_\Box) = \Box$ immediately, and that if $s \in S_{\mathbf{c}[\mathbf{1}]}$, then $s \in S_\mathbf{c}$, and hence $\lambda_{\mathbf{c}[\mathbf{1}]}(s) = \lambda_\mathbf{c}(s)$.

Conversely, we can show that $\mathbf{c}[\mathbf{1}]$ is isomorphic to $\mathbf{u}$ satisfying the above conditions, and hence that $C[1] = U$. In detail, first note that $S_{\mathbf{c}[\mathbf{1}]} = S_\mathbf{c} \cup S_\mathbf{1} - \{s_\Box\} = S_\mathbf{c} - \{s_\Box\} = S_\mathbf{u}$ by the first property; we choose the identity on $S_\mathbf{c}$ to be the mediating isomorphism. To see that this indeed gives us a labelled poset isomorphism, note that the identity preserves and reflects ordering by the first property, and it preserves labels by the second property. $\Box$

Lemma 5.49. *Let $C \in \mathsf{PC}$ and $U \in \mathsf{SP}$ such that $C[1] = U$. The following hold:*

(i) We can construct a $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C'[1] = U$ and $C \sqsubseteq C'$.

(ii) We can construct a $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C'[1] = U$ and $C' \sqsubseteq C$.

Proof. We prove the second claim; the first claim can be shown using a similar argument. Let $C = [\mathbf{c}]$ and $U = [\mathbf{u}]$. Without loss of generality, assume that $\mathbf{c}[\mathbf{1}] = \mathbf{u}$. We will show that if $C \notin \mathsf{PC}^{\mathsf{sp}}$, then we can construct a labelled poset $\mathbf{c}'$ such that all of the following hold:

(a) $S_\mathbf{c} = S_{\mathbf{c}'}$, and

(b) for all $s, s' \in S_\mathbf{c} \setminus \{s_\Box\}$ with $s \leq_{\mathbf{c}'} s'$, we have $s \leq_\mathbf{c} s'$, and

(c) $\lambda_\mathbf{c} = \lambda_{\mathbf{c}'}$, and

(d) $\leq_\mathbf{c}$ is contained in but not equal to $\leq_{\mathbf{c}'}$.

Conditions (a)–(c), in combination with Lemma 5.B.2, imply that $C[1] = [\mathbf{c}'[\mathbf{1}]]$. Moreover, (a), (c) and (d) tell us that $[\mathbf{c}'] \sqsubseteq [\mathbf{c}]$ but $[\mathbf{c}'] \neq [\mathbf{c}]$. Hence, $[\mathbf{c}']$ is strictly subsumed by C but still satisfies the premise of the lemma. By well-founded induction on $\sqsubseteq$, we can repeat this process until we find a context $C' \in \mathsf{PC}^{\mathsf{sp}}$ that is subsumed by C and still satisfies that $C'[1] = U$.

Recall that an N-shape in $\mathbf{v}$ is a quadruplet of events $\langle s_1, s_2, s_3, s_4 \rangle \in S_\mathbf{v}^4$ such that:

$$s_1 \leq_\mathbf{v} s_3 \qquad s_2 \leq_\mathbf{v} s_3 \qquad s_2 \leq_\mathbf{v} s_4 \qquad s_1 \not\leq_\mathbf{v} s_4 \qquad s_2 \not\leq_\mathbf{v} s_1 \qquad s_4 \not\leq_\mathbf{v} s_3.$$

Since $C \notin \mathsf{PC}^{\mathsf{sp}}$ but $C \in \mathsf{PC}$, it follows by Theorem 3.22 and Lemma 5.47 that C is not series-parallel; hence, there is an N-shape $\langle s_1, s_2, s_3, s_4\rangle \in S_{\mathbf{c}}^4$. On the other hand, since $U = C[1] \in \mathsf{SP}$ we know that U does not have this pattern, so $s_\square$ must be one of these four events. Let us do a case analysis:

1. First, suppose that $s_\square = s_1$. We claim that we can build $\mathbf{c}'$ by choosing

   $$S_{\mathbf{c}'} = S_{\mathbf{c}} \qquad \lambda_{\mathbf{c}'} = \lambda_{\mathbf{c}} \qquad \leq_{\mathbf{c}'} = (\leq_{\mathbf{c}} \cup \{\langle s_\square, s_4\rangle\})^*$$

   Conditions (a), (c) and (d) hold by construction, and $\leq_{\mathbf{c}'}$ is reflexive and transitive, too. It remains to validate condition (b), and that $\leq_{\mathbf{c}'}$ is antisymmetric. The following facts help.

   Fact 5.B.3. *For all $s \in S_{\mathbf{c}}$ with $s <_{\mathbf{c}} s_\square$ we have $s \leq_{\mathbf{c}} s_4$.*

   Proof of Fact 5.B.3. The proof proceeds by contradiction: assume there exists $s \in S_{\mathbf{c}}$ with $s <_{\mathbf{c}} s_\square$ and $s \not\leq_{\mathbf{c}} s_4$. Then we can show that the quadruplet $\langle s, s_2, s_3, s_4\rangle \in S_{\mathbf{c}} \setminus \{s_\square\} = S_{\mathbf{u}}$ is an N-shape in $\mathbf{u}$. Indeed, we already know that

   $$s_2 \leq_{\mathbf{u}} s_3 \qquad s_2 \leq_{\mathbf{u}} s_4 \qquad s_4 \not\leq_{\mathbf{u}} s_3.$$

   Therefore what remains are the statements relating to s, i.e., that

   $$s \leq_{\mathbf{u}} s_3 \qquad s \not\leq_{\mathbf{u}} s_4 \qquad s_2 \not\leq_{\mathbf{u}} s$$

   The first one is obtained by transitivity: $s \leq_{\mathbf{c}} s_\square \leq_{\mathbf{c}} s_3$, and thus $s \leq_{\mathbf{u}} s_3$. The second one is a direct consequence of our assumption that $s \not\leq_{\mathbf{c}} s_4$. Lastly, if we assume $s_2 \leq_{\mathbf{u}} s$, then $s_2 \leq_{\mathbf{c}} s$, and by transitivity we get $s_2 \leq_{\mathbf{c}} s_\square$, which contradicts that $(s_\square, s_2, s_3, s_4)$ is an N-shape. We now have shown that $s \not\leq_{\mathbf{c}} s_4$ implies the existence of an N-shape in U, which cannot be the case. We conclude that if $s <_{\mathbf{c}} s_\square$, then $s \leq_{\mathbf{c}} s_4$. □

   Fact 5.B.4. *If $s \leq_{\mathbf{c}'} s'$, then either $s \leq_{\mathbf{c}} s'$, or $s = s_\square$ and $s_4 \leq_{\mathbf{c}} s'$.*

   Proof of Fact 5.B.4. We shall phrase the claim and its proof in terms of relations. Let $R = \{\langle s_\square, s_4\rangle\}$ and $T = <_{\mathbf{c}}$; note that $\leq_{\mathbf{c}} = (T \cup R)^*$. It now suffices to show that

   $$(T \cup R)^* \subseteq T^* \cup R \circ T^*$$

   Note that Fact 5.B.3 can be written as $T \circ R \subseteq T^*$. Also, $R \circ R = \emptyset$, since $s_\square \neq s_4$, because $s_\square \leq_{\mathbf{c}} s_3$ and $s_4 \not\leq_{\mathbf{u}} s_3$. Using these properties, we can derive the following containments:

   $$T \circ T^* \subseteq T^* \cup R \circ T^* \qquad T \circ R \circ T^* \subseteq T^* \circ T^* \subseteq T^* \cup R \circ T^*$$

   $$R \circ T^* \subseteq T^* \cup R \circ T^* \qquad R \circ R \circ T^* = \emptyset \circ T^* = \emptyset \subseteq T^* \cup R \circ T^*$$

By distributivity of relational composition over union, we can then derive the following:

$$(T \cup R) \circ (T^* \cup R \circ T^*) = T \circ T^* \cup T \circ R \circ T^* \cup R \circ T^* \cup R \circ R \circ T^* \subseteq T^* \cup R \circ T^*.$$

By the fixpoint principle for reflexive-transitive closure, it follows that:

$$(T \cup R)^* \circ (T^* \cup R \circ T^*) \subseteq T^* \cup R \circ T^*.$$

Finally, we conclude that $(T \cup R)^* \subseteq T^* \cup R \circ T^*$ by:

$$(T \cup R)^* \subseteq (T \cup R)^* \circ (T^* \cup R \circ T^*) \subseteq T^* \cup R \circ T^* \qquad \square$$

We can now use Fact 5.B.4 to show the remaining properties. For the third condition on $\mathbf{c}'$, assume $s \leq_{\mathbf{c}'} s'$ with $s, s' \neq s_\square$. By the previous observation, either $s \leq_{\mathbf{c}} s'$ or we have both $s = s_\square$ and $s_4 \leq_{\mathbf{c}} s'$. Since $s \neq s_\square$, we may conclude that $s \leq_{\mathbf{c}} s'$.

For antisymmetry, let $s \leq_{\mathbf{c}'} s' \leq_{\mathbf{c}'} s$. Using Fact 5.B.4, we distinguish four cases:

(a) If $s \leq_{\mathbf{c}} s' \leq_{\mathbf{c}} s$, then by antisymmetry of $\leq_{\mathbf{c}}$ we get $s = s'$;

(b) If $s \leq_{\mathbf{c}} s'$ and $s' = s_\square$ with $s_4 \leq_{\mathbf{c}} s$, then we get $s_4 \leq_{\mathbf{c}} s \leq_{\mathbf{c}} s' = s_\square$;

(c) If $s = s_\square$ with $s_4 \leq_{\mathbf{c}} s'$ and $s' \leq_{\mathbf{c}} s$, then we get $s_4 \leq_{\mathbf{c}} s' \leq_{\mathbf{c}} s = s_\square$;

(d) If $s = s_\square$ with $s_4 \leq_{\mathbf{c}} s'$ and $s' = s_\square$ with $s_4 \leq_{\mathbf{c}} s$, then we get $s = s_\square = s'$.

In the first and last case we could conclude that $s = s'$, while in the other three we ended up with $s_4 \leq_{\mathbf{c}} s_\square$, contradicting that $s_\square \leq_{\mathbf{c}} s_3$ and $s_4 \not\leq_{\mathbf{u}} s_3$.

2. Next, suppose that $s_\square = s_2$. We claim that we can build $\mathbf{c}'$ by choosing

$$S_{\mathbf{c}'} = S_{\mathbf{c}} \qquad \lambda_{\mathbf{c}'} = \lambda_{\mathbf{c}} \qquad \leq_{\mathbf{c}'} = (\leq_{\mathbf{c}} \cup \{\langle s_\square, s_1 \rangle\})^*$$

As before, conditions (a), (c) and (d) hold by construction, and $\leq_{\mathbf{c}'}$ is reflexive and transitive. It remains to validate (b), and that $\leq_{\mathbf{c}'}$ is antisymmetric. The following facts help.

Fact 5.B.5. *For all $s \in S_{\mathbf{c}}$ with $s <_{\mathbf{c}} s_\square$, we have $s \leq_{\mathbf{c}} s_1$.*

Proof of Fact 5.B.5. We proceed by contradiction. Assume that $s <_{\mathbf{c}} s_\square$ and $s \not\leq_{\mathbf{c}} s_1$ Then we show that $\langle s_1, s, s_3, s_4 \rangle \in S_{\mathbf{c}} \setminus \{s_\square\} = S_{\mathbf{u}}$ is an N-shape. Indeed, we already know that

$$s_1 \leq_{\mathbf{u}} s_3 \qquad s_1 \not\leq_{\mathbf{u}} s_4 \qquad s_4 \not\leq_{\mathbf{u}} s_3.$$

Therefore what remains are the statements relating to s, i.e.,

$$s \leq_{\mathbf{u}} s_3 \qquad s \leq_{\mathbf{u}} s_4 \qquad s \not\leq_{\mathbf{u}} s_1$$

The first and second are obtained by transitivity: $s \leq_{\mathbf{c}} s_\square \leq_{\mathbf{c}} s_3, s_4$. The third one follows directly from our assumption. We now have shown that $s \not\leq_{\mathbf{c}} s_1$ implies the existence of an N-pattern in U, which cannot be the case. We conclude that if $s <_{\mathbf{c}} s_\square$, then $s \leq_{\mathbf{c}} s_1$. □

Fact 5.B.6. *If $s \leq_{\mathbf{c}'} s'$, then either $s \leq_{\mathbf{c}} s'$ or $s = s_\square$ and $s_1 \leq_{\mathbf{c}} s'$.*

Proof of Fact 5.B.6. As before, we formulate and prove claim in terms of relations. Let $R = \{\langle s_\square, s_1\rangle\}$ and $T = <_{\mathbf{c}}$. We have $\leq_{\mathbf{c}} = (T \cup R)^*$, which makes the claim equivalent to

$$(T \cup R)^* \subseteq T^* \cup R \circ T^*$$

We can write Fact 5.B.5 as $T \circ R \subseteq T^*$. Note also that $T \circ T = \emptyset$, since $s_\square \neq s_1$. Since we now have the same premises as in Fact 5.B.4, we may derive the same conclusion. □

The desired properties now follow from Fact 5.B.6, as in the previous case.

3. If $s_\square = s_3$, then by a similar argument as in the case where $s_\square = s_2$, we may show first that for all s with $s_\square <_{\mathbf{c}} s$ it holds that $s_4 <_{\mathbf{c}} s$. We can then use this to show that choosing $\leq_{\mathbf{c}'} = (\leq_{\mathbf{c}} \cup \{\langle s_4, s_\square\rangle\})^*$ validates the claim.
4. If $s_\square = s_4$, then by a similar argument as in the case where $s_\square = s_1$, we may show first that for all s with $s_\square <_{\mathbf{c}} s$ it holds that $s_1 <_{\mathbf{c}} s$. We can then use this to show that choosing $\leq_{\mathbf{c}'} = (\leq_{\mathbf{c}} \cup \{\langle s_1, s_\square\rangle\})^*$ validates the claim. □

Lemma 5.50. *Let $C \in \mathsf{PC}$ and $V \in \mathsf{Pom}$. The following hold:*

(i) If $C[1] \sqsubseteq V$, then we can construct $C' \in \mathsf{PC}$ such that $C \sqsubseteq C'$ and $C'[1] = V$.

(ii) If $V \sqsubseteq C[1]$, then we can construct $C' \in \mathsf{PC}$ such that $C' \sqsubseteq C$ and $C'[1] = V$.

Moreover, if $V \in \mathsf{SP}$, then the C' constructed in each of the above is series-parallel.

Proof. Let $U = C[1]$. It suffices to construct a $C' \in \mathsf{PC}$ such that $C' \sqsubseteq C$ and $C'[1] = V$, since Lemma 5.49 takes care of the "moreover" clause. By Lemma 5.B.2, we find that $C = [\mathbf{c}]$ and $U = [\mathbf{u}]$ such that $S_{\mathbf{c}} = S_{\mathbf{u}} \cup \{s_\square\}$, for all $s, s' \in S_{\mathbf{u}}$ it holds that $s \leq_{\mathbf{u}} s'$ if and only if $s \leq_{\mathbf{c}} s'$, $\lambda_{\mathbf{c}}(s_\square) = \square$ and $\lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s)$ for all $s \in S_{\mathbf{u}}$. Let $V = [\mathbf{v}]$; since $V \sqsubseteq U$, we know without loss of generality that $S_{\mathbf{v}} = S_{\mathbf{u}}$ and $\lambda_{\mathbf{u}} = \lambda_{\mathbf{v}}$ and $\leq_{\mathbf{u}} \subseteq \leq_{\mathbf{v}}$. Let $\leq_{\mathbf{c}'}$ be the smallest transitive relation on $S_{\mathbf{c}}$ containing $\leq_{\mathbf{v}}$ and $\leq_{\mathbf{c}}$. Let $s, s' \in S_{\mathbf{c}}$ with $s \leq_{\mathbf{c}'} s'$; the following properties of $\leq_{\mathbf{c}'}$ are useful:

(a) If $s = s_\square$ and $s' \in S_{\mathbf{v}}$, then there exists an $\hat{s} \in S_{\mathbf{v}}$ such that $s_\square \leq_{\mathbf{c}} \hat{s} \leq_{\mathbf{v}} s'$.

(b) If $s \in S_{\mathbf{v}}$ and $s' = s_{\Box}$, then there exists an $\hat{s} \in S_{\mathbf{v}}$ such that $s \leq_{\mathbf{v}} \hat{s} \leq_{\mathbf{c}} s_{\Box}$.

(c) If $s, s' \in S_{\mathbf{v}}$, then $s \leq_{\mathbf{v}} s'$.

We prove these claims in tandem by induction on the construction of $\leq_{\mathbf{c}'}$. In the base, suppose for the first claim that $s = s_{\Box}$ and $s' \in S_{\mathbf{v}}$; we then know that $s \leq_{\mathbf{c}} s'$ (the case where $s \leq_{\mathbf{v}} s'$ can be excluded, for $s_{\Box} \notin S_{\mathbf{v}}$), and hence we can choose $\hat{s} = s'$ to satisfy the claim; the second claim goes through similarly. For the last claim, if $s \leq_{\mathbf{v}} s'$ then we are done immediately; otherwise, if $s \leq_{\mathbf{c}} s'$, then since $s, s' \neq s_{\Box}$ we have that $s \leq_{\mathbf{u}} s'$, and hence $s \leq_{\mathbf{v}} s'$.

In the inductive step, we have that $s \leq_{\mathbf{c}'} s'$ because there exists an $s'' \in S_{\mathbf{c}}$ with $s \leq_{\mathbf{c}'} s'' \leq_{\mathbf{c}'} s'$. We shall treat each claim separately, assuming that all three claims hold for $s \leq_{\mathbf{c}'} s''$ and $s'' \leq_{\mathbf{c}'} s'$.

(a) If $s = s_{\Box}$ and $s' \in S_{\mathbf{v}}$, then there are two cases to consider. On the one hand, if $s'' = s_{\Box}$, then we can apply the induction hypothesis to $s'' \leq_{\mathbf{c}'} s'$ to find an $\hat{s} \in S_{\mathbf{v}}$ such that $s_{\Box} \leq_{\mathbf{c}} \hat{s} \leq_{\mathbf{v}} s'$.

On the other hand, if $s'' \neq s_{\Box}$, then $s'' \in S_{\mathbf{v}}$, and we can apply the induction hypothesis to $s \leq_{\mathbf{c}'} s''$ to find an $\hat{s} \in S_{\mathbf{v}}$ such that $s_{\Box} \leq_{\mathbf{c}} \hat{s} \leq_{\mathbf{v}} s''$. By applying the induction hypothesis to $s'' \leq_{\mathbf{c}'} s'$, we find $s'' \leq_{\mathbf{v}} s'$, and thus we can conclude that $s_{\Box} \leq_{\mathbf{c}} \hat{s} \leq_{\mathbf{v}} s'$.

(b) If $s \in S_{\mathbf{v}}$ and $s = s_{\Box}$, then the proof proceeds as in the previous case.

(c) If $s, s' \in S_{\mathbf{v}}$, then there are again two cases to consider.

- If $s'' = s_{\Box}$, then we can apply the induction hypothesis (specifically, the second claim) to $s \leq_{\mathbf{c}'} s''$ to find an $\hat{s} \in S_{\mathbf{v}}$ such that $s \leq_{\mathbf{v}} \hat{s} \leq_{\mathbf{c}} s''$. Similarly, we can apply the induction hypothesis (in this case, the first claim) to $s'' \leq_{\mathbf{c}'} s$ to find an $\hat{s}' \in S_{\mathbf{v}}$ such that $s'' \leq_{\mathbf{c}} \hat{s}' \leq_{\mathbf{v}} s'$. We then know that $\hat{s} \leq_{\mathbf{c}} \hat{s}'$, and since $\hat{s}, \hat{s}' \in S_{\mathbf{v}} = S_{\mathbf{u}}$, we know that $\hat{s} \leq_{\mathbf{u}} \hat{s}'$, and hence $\hat{s} \leq_{\mathbf{v}} \hat{s}'$. In total, we find that $s \leq_{\mathbf{v}} \hat{s} \leq_{\mathbf{v}} \hat{s}' \leq_{\mathbf{v}} s'$.
- If $s'' \neq s_{\Box}$, then $s'' \in S_{\mathbf{v}}$, and we can apply the induction hypothesis to both $s \leq_{\mathbf{c}'} s''$ and $s'' \leq_{\mathbf{c}'} s'$ to find that $s \leq_{\mathbf{v}} s'' \leq_{\mathbf{v}} s'$.

We now claim that $\leq_{\mathbf{c}'}$ is antisymmetric. To see this, suppose that $s, s' \in S_{\mathbf{c}}$ with $s \leq_{\mathbf{c}'} s' \leq_{\mathbf{c}'} s$. Now, if $s, s' \in S_{\mathbf{v}}$, then $s \leq_{\mathbf{v}} s' \leq_{\mathbf{v}} s$ by property (c), and hence $s = s'$ by antisymmetry of $\leq_{\mathbf{v}}$. Otherwise, if $s = s_{\Box}$, then suppose towards a contradiction that $s' \neq s_{\Box}$; in that case, $s' \in S_{\mathbf{v}}$, and we can find $\hat{s}, \hat{s}' \in S_{\mathbf{v}}$ such that $s_{\Box} \leq_{\mathbf{c}} \hat{s} \leq_{\mathbf{v}} s' \leq_{\mathbf{v}} \hat{s}' \leq_{\mathbf{c}} s_{\Box}$ by properties (a) and (b). But then, since $\hat{s}' \leq_{\mathbf{c}} \hat{s}$, it follows that $\hat{s}' \leq_{\mathbf{v}} \hat{s}$. Moreover, $\hat{s} \leq_{\mathbf{v}} s' \leq_{\mathbf{v}} \hat{s}' \leq \hat{s}$, and hence $\hat{s}' = s' = \hat{s}$ by antisymmetry of $\leq_{\mathbf{v}}$. It then follows that $s_{\Box} \leq_{\mathbf{c}} s' \leq_{\mathbf{c}} s_{\Box}$, meaning that $s' = s_{\Box}$ by antisymmetry of $\leq_{\mathbf{c}}$ — a contradiction. We conclude that $s' = s_{\Box} = s$.

Since $\leq_{\mathbf{c}'}$ is reflexive and transitive by construction, and antisymmetric by the above, it is a partial order. We now choose $S_{\mathbf{c}'} = S_{\mathbf{c}}$ and $\lambda_{\mathbf{c}'} = \lambda_{\mathbf{c}}$, and let $C' = [\mathbf{c}']$. Note that C' has exactly one $s_\Box$-labelled node, and hence $C' \in \mathsf{PC}$. Now, if $s, s' \in S_{\mathbf{v}}$, then $s \leq_{\mathbf{v}} s'$ implies $s \leq_{\mathbf{c}'} s'$ by definition of $\leq_{\mathbf{c}'}$; furthermore, if $s \leq_{\mathbf{c}'} s'$, then $s \leq_{\mathbf{v}} s'$ by property (c) above. Since $S_{\mathbf{c}'} = S_{\mathbf{c}} = S_{\mathbf{u}} \cup \{s_\Box\} = S_{\mathbf{v}} \cup \{s_\Box\}$, and furthermore $\lambda_{\mathbf{c}'}(s_\Box) = \lambda_{\mathbf{c}}(s_\Box) = \Box$ and $\lambda_{\mathbf{c}'}(s) = \lambda_{\mathbf{c}}(s) = \lambda_{\mathbf{u}}(s) = \lambda_{\mathbf{v}}(s)$ for $s \in S_{\mathbf{v}}$, we have that $C'[1] = V$ by Lemma 5.B.2. Lastly, $\leq_{\mathbf{c}} \subseteq \leq_{\mathbf{c}'}$, and thus $C' \sqsubseteq C$. □

Lemma 5.51. *Let H be a set of hypotheses, let $L \subseteq \mathsf{SP}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$. The following hold:*

(i) If H is right-simple and $e \leq f \in H$ such that $C[\![f]\!] \subseteq (L^H)^{\mathsf{exch}}$, then $C[\![e]\!] \subseteq (L^H)^{\mathsf{exch}}$.

(ii) If H is left-simple and $e \leq f \in \mathsf{exch}$ such that $C[\![f]\!] \subseteq (L^{\mathsf{exch}})^H$, then $C[\![e]\!] \subseteq (L^{\mathsf{exch}})^H$.

Proof. We treat the claims in the order given.

(i) We know that, if $e \leq f \in H$, then $f = 1$ or $f = \mathsf{a}$ for some $\mathsf{a} \in \Sigma$. We denote the sole pomset in $[\![f]\!]$ by U; observe that either $U = 1$ or $U = \mathsf{a}$ for some a. Using Corollary 5.6, it then suffices to show that if $C \in \mathsf{PC}^{\mathsf{sp}}$ and $C[\mathsf{a}] \sqsubseteq^{\mathsf{sp}} V \in L^H$, then $C[\![e]\!] \subseteq (L^H)^{\mathsf{exch}}$.

Given the premises above and applying Lemma 5.50(i) as well as Lemma 5.48(i), we can find $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C \sqsubseteq C'$ and $C'[U_f] = V$. But then, by definition of closure, it follows that $C'[\![e]\!] \subseteq L^H$. Now, let $W \in C[\![e]\!]$; then $W = C[X]$ for some $X \in [\![e]\!]$, which means that $W \sqsubseteq C'[X]$. Since $C'[X] \in C'[\![e]\!] \subseteq L^H$, it follows that $W \in (L^H)^{\mathsf{exch}}$ by Corollary 5.6.

(ii) In this case, Corollary 5.6 tells us that the claim is equivalent to showing that if $U \sqsubseteq^{\mathsf{sp}} V \in (L^{\mathsf{exch}})^H$, then $U \in (L^{\mathsf{exch}})^H$, or, more generally, that if $U \sqsubseteq^{\mathsf{sp}} V \in B \subseteq (L^{\mathsf{exch}})^H$, then $U \in (L^{\mathsf{exch}})^H$. Our proof proceeds by induction on the construction of $B \subseteq (L^{\mathsf{exch}})^H$. In the base, we have that $B = L^{\mathsf{exch}}$, in which case $U \in L^{\mathsf{exch}} \subseteq (L^{\mathsf{exch}})^H$ immediately. For the inductive step, we obtain $e \leq f \in H$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ such that $B = C[\![e]\!]$ and $C[\![f]\!] \subseteq (L^{\mathsf{exch}})^H$. Since H is left-simple, we know that $e = 1$ or $e = \mathsf{a}$. Let us write X for the sole pomset in $[\![e]\!]$, and observe that $X = 1$ or $X = \mathsf{a}$ for some $\mathsf{a} \in \Sigma$. We then know that $V = C[X]$. By Lemma 5.50(ii) and Lemma 5.48(ii), we find $C' \in \mathsf{PC}^{\mathsf{sp}}$ such that $C' \sqsubseteq C$ and $C'[X] = U$. Since very pomset in $C'[\![f]\!]$ is subsumed by one in $C[\![f]\!]$, we find by induction that $C'[\![f]\!] \subseteq (L^{\mathsf{exch}})^H$, and thus $U = C'[X] \in (L^{\mathsf{exch}})^H$ by definition of closure. □

Theorem 5.52. *Let H be a set of hypotheses. The following hold.*

(i) If H is right-simple, then $H \cup \mathsf{exch}$ factorises into H, exch.

(ii) If H is left-simple, then $H \cup \mathsf{exch}$ factorises into exch, H.

Proof. We prove the first claim; the second claim follows by a similar argument. First, note that since $H \cup \mathsf{exch}$ implies H, we have that $L^H \subseteq L^{H\cup\mathsf{exch}}$ by Lemma 4.23, and hence $(L^H)^{\mathsf{exch}} \subseteq (L^{H\cup\mathsf{exch}})^{H\cup\mathsf{exch}} = L^{H\cup\mathsf{exch}}$, where the latter equality follows from Lemma 4.11.

For the other inclusion, we show that if $A \subseteq L^{H\cup\mathsf{exch}}$, then $A \subseteq (L^H)^{\mathsf{exch}}$, by induction on $A \subseteq L^{H\cup\mathsf{exch}}$. In the base, where $A = L$, note that $A = L \subseteq L^H \subseteq (L^H)^{\mathsf{exch}}$. For the inductive step, we obtain $e \leq f \in H \cup \mathsf{exch}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ such that $A = C[\llbracket e \rrbracket]$ and $C[\llbracket f \rrbracket] \subseteq L^{H\cup\mathsf{exch}}$. By induction, it then follows that $C[\llbracket f \rrbracket] \subseteq (L^H)^{\mathsf{exch}}$. There are now two cases to consider.

- If $e \leq f \in H$, then because H is right-simple, we have that $C[\llbracket e \rrbracket] \subseteq (L^H)^{\mathsf{exch}}$ by Lemma 5.51.
- On the other hand, if $e \leq f \in \mathsf{exch}$, then $A = C[\llbracket e \rrbracket] \subseteq (L^H)^{\mathsf{exch}}$ by definition of closure. □

Chapter 6

Control Flow

Thus far, we have considered macroscopic program composition: run either this or that program, run this before that other piece of code, and have these threads execute in parallel. Our toolkit lacks the ability of a program to act on the current state: do this if that is true, or do this while that holds. If we had this type of composition available in our syntax and axioms, we could perhaps show that some complicated conditional can be simplified, or we could prove that a certain program with a doubly nested loop is equivalent to a program with a single loop.

For rational expressions, i.e., series-rational expressions without parallel composition, an extension along these lines was proposed and studied by Kozen and collaborators, in the form of *Kleene algebra with tests (KAT)* [Koz96; CKS96; KS96; Koz97]; Jipsen and Moshier studied the extension of these techniques to sr-expressions [JM16]. For our purposes, we can think of the relevant (series-)rational expressions over an extended alphabet, where some letters represent the actions available to a program, and the other letters represent *assertions* about the state. These assertions furthermore have an internal algebraic structure: if p and q are assertions, then so is $p \wedge q$, i.e., the assertion that both p and q are true, as is $\overline{p}$, i.e., the assertion that p is false.

One can then use these assertions to embed structures that model control flow. For instance, a conditional like **if** p **then** e **else** f for programs e and f and an assertion p can be encoded as $p \cdot e + \overline{p} \cdot f$: either p holds, after which e is executed, or p does not hold, in which case we run f. Similarly, we can encode a loop like **while** p **do** e, where p is an assertion and e represents a program, by $(p \cdot e)^* \cdot \overline{p}$: run e some number of times, where before each iteration p is true, such that in the end p no longer holds. Assertions also come with reasoning rules, and we can use these to reason about control flow [Koz97; AK01; KP00; BK02].

Our objective in this chapter is to study the inclusion of reasoning about control flow to concurrent programs, using the tools derived in previous chapters. There are two main contributions. We shall show that, on the one hand, combining existing hypotheses to encode control flow with the exchange law can lead to a system that is not very useful for reasoning about program equivalence. On the other hand, we show that these hypotheses can be slightly weakened to obtain a system that does not have this problem, and for which we can derive completeness and decidability.

We start by casting KAT in the framework of hypotheses, starting with the syntax.

Definition 6.1 (Syntax). Let Ω be a (finite) alphabet of symbols called *tests*. The set of *propositional expressions* is denoted $\mathcal{T}_{\mathsf{B}}$, and generated by the grammar

$$p, q ::= \bot \mid \top \mid o \in \Omega \mid p \vee q \mid p \wedge q \mid \overline{p}$$

The set of *guarded rational expressions*, denoted $\mathcal{T}_{\mathsf{G}}$, contains rational expressions over $\Sigma \cup \mathcal{T}_{\mathsf{B}}$; i.e., generated by

$$e, f ::= 0 \mid 1 \mid \mathsf{a} \in \Sigma \mid p \in \mathcal{T}_{\mathsf{B}} \mid e + f \mid e \cdot f \mid e^*$$

In the syntax for propositional expressions, the symbol $\bot$ represents the assertion that is always false, $\top$ is the assertion that is always true, $p \vee q$ asserts that either p or q holds, $p \wedge q$ asserts that both p and q hold, and $\overline{p}$ is the assertion that succeeds precisely when p is false.

Having defined how to embed assertions in rational expressions, we now need a way to reason about them. The first and most obvious rule is that assertions satisfy the axioms of Boolean algebra, e.g., asserting that p or $\overline{p}$ hold is the same as the assertion that is always true. Next, we connect the assertions to the syntax of rational expressions themselves: for instance, asserting that p holds and then immediately asserting that q holds should be the same as asserting that p and q hold at the same time. Similarly, running the assertion $\top$, which always succeeds, has the same behaviour as 1, the program that does nothing. In total, we arrive at the following hypotheses.

Definition 6.2 (Axioms). We define $\equiv_{\mathsf{B}}$ as the smallest congruence on $\mathcal{T}_{\mathsf{B}}$ that satisfies

$$p \vee \bot \equiv_{\mathsf{B}} p \qquad p \vee q \equiv_{\mathsf{B}} q \vee p \qquad p \vee \overline{p} \equiv_{\mathsf{B}} \top \qquad p \vee (q \vee r) \equiv_{\mathsf{B}} (p \vee q) \vee r$$

$$p \wedge \top \equiv_{\mathsf{B}} p \qquad p \wedge q \equiv_{\mathsf{B}} q \wedge p \qquad p \wedge \overline{p} \equiv_{\mathsf{B}} \bot \qquad p \wedge (q \wedge r) \equiv_{\mathsf{B}} (p \wedge q) \wedge r$$

$$p \vee (q \wedge r) \equiv_{\mathsf{B}} (p \vee q) \wedge (p \vee r) \qquad p \wedge (q \vee r) \equiv_{\mathsf{B}} (p \wedge q) \vee (p \wedge r)$$

We define kat as the union of the following sets of hypotheses

$$\mathsf{bool} = \{p = q \ : \ p \equiv_{\mathsf{B}} q\} \qquad \mathsf{unit} = \{0 = \bot, 1 = \top\}$$

$$\mathsf{choice} = \{p + q = p \vee q \ : \ p, q \in \mathcal{T}_{\mathsf{B}}\} \qquad \mathsf{seq} = \{p \cdot q = p \wedge q \ : \ p, q \in \mathcal{T}_{\mathsf{B}}\}$$

Convention 6.3. Because conjunction and disjunction are commutative and associative (up to $\equiv_{\mathsf{B}}$), the bracketing and ordering of an expression like $p_1 \vee \cdots \vee p_n$ is unimportant as far as $\equiv_{\mathsf{B}}$ is concerned. For a finite $S = \{p_1, \ldots, p_n\} \subseteq \mathcal{T}_{\mathsf{B}}$, this allows us to unambiguously (up to $\equiv_{\mathsf{B}}$) denote

$$\bigvee S = p_1 \vee \cdots \vee p_1 \qquad\qquad \bigwedge S = p_1 \wedge \cdots \wedge p_n$$

Remark 6.4. The usual syntax for KAT elides $\bot$ and $\top$, identifying them with 0 and 1 at the syntactic level; sequential composition of tests is similarly identified with conjunction, and non-deterministic composition of tests is syntactically the same as disjunction. It is not very hard to show that, with these identifications, $\equiv_{\mathsf{R}}^{\mathsf{kat}}$ coincides with the usual congruence used in Kleene algebra with tests, and that $[\![-]\!]^{\langle \mathsf{kat} \rangle}$ is isomorphic to the traditional semantics of KAT given in terms of *guarded strings*, i.e., strings in which letters and atoms of the Boolean algebra alternate [KS96].

6.1 The perils of concurrent KAT

We can extend the syntax of series-rational expressions with assertions analogously.

Definition 6.5. The set of *guarded series-rational expressions* (or *gsr-expressions*, for short), denoted $\mathcal{T}_{\mathsf{GSR}}$, is formed by series-rational expressions over $\Sigma \cup \mathcal{T}_{\mathsf{B}}$; alternatively, it is generated by

$$e, f ::= 0 \;\mid\; 1 \;\mid\; \mathsf{a} \in \Sigma \;\mid\; p \in \mathcal{T}_{\mathsf{B}} \;\mid\; e + f \;\mid\; e \cdot f \;\mid\; e \parallel f \;\mid\; e^*$$

The most obvious way forward, then, is to take the hypotheses in kat as well as exch and use them to reason about gsr-expressions [JM16]. The problem is that such a system would not make a lot of sense when applied to concurrent programs. To see this, consider the following observation.

Fact 6.6. *Let* $\mathsf{ckat} = \mathsf{kat} \cup \mathsf{exch}$. *If* $e \in \mathcal{T}_{\mathsf{GSR}}$ *and* $p \in \mathcal{T}_{\mathsf{B}}$, *then* $p \cdot e \cdot \overline{p} \equiv^{\mathsf{ckat}} 0$.

Proof. Using the hypotheses in exch, we first derive that

$$p \cdot e \cdot \overline{p} \leqq^{\mathsf{exch}} (p \parallel e) \cdot \overline{p} \equiv (p \parallel e) \cdot (\overline{p} \parallel 1) \leqq^{\mathsf{exch}} (p \cdot \overline{p}) \parallel (e \cdot 1) \equiv (p \cdot \overline{p}) \parallel e$$

Using the hypotheses in kat, we next derive that

$$(p \cdot \overline{p}) \parallel e \equiv^{\mathsf{kat}} (p \wedge \overline{p}) \parallel e \equiv^{\mathsf{kat}} \bot \parallel e \equiv^{\mathsf{kat}} 0 \parallel e \equiv 0 \qquad \square$$

The above invalidates the use of $\equiv^{\mathsf{ckat}}$ as a tool to reason about program equivalence. It says that first asserting that p is true, running a program e, and then asserting that p is false, is equivalent to the program 0 (i.e., the program without valid behaviour) for *any* assertion p and *any* program e.

In other words, there is no way for e to affect the outcome of p — all assertions are invariants of all programs! This has further ramifications for encoding control flow; for instance, a corollary of the above is that $(p \cdot e)^* \cdot \overline{p}$, the encoding of **while** p **do** e, is equivalent to $\overline{p}$, since we can derive that

$$(p \cdot e)^* \cdot \overline{p} \equiv (1 + (p \cdot e)^* \cdot p \cdot e) \cdot \overline{p} \equiv \overline{p} + (p \cdot e)^* \cdot p \cdot e \cdot \overline{p} \equiv^{\mathsf{ckat}} \overline{p} + (p \cdot e)^* \cdot 0 \equiv \overline{p}$$

In other words, the body of any loop does not affect the program at all.

Conceptually, one could argue that the mistake in trying to lift kat to sr-expressions is that we included the hypotheses $\{p \cdot q \leq p \wedge q \ : \ p, q \in \mathcal{T}_\mathsf{B}\}$. For sequential programs, these make sense: the behaviour of asserting that p and q hold at the same time contains the behaviour of asserting that p is true, and then q is true, because nothing happens in the interim. With concurrency, such hypotheses cause trouble. For instance, if $p \cdot \overline{p}$ runs in parallel with e, actions from e may be interleaved between p and $\overline{p}$ to make sure that both are true in succession, even though they cannot be true at the same time [CHM17]. This happened in the derivation for Fact 6.6 above.

Remark 6.7. Jipsen and Moshier [JM16] proposed a set of laws on gsr-expressions similar to those of KAT. Even though their hypotheses on the propositional expressions are those of a *pseudo-complemented distributive lattice* [BD74] and therefore strictly weaker than Boolean algebra, $p \wedge \overline{p}$ remains equivalent to $\bot$ in their system, and hence the derivation in Fact 6.6 still applies.

Since the hypotheses $\{p \cdot q \leq p \wedge q \ : \ p, q \in \mathcal{T}\}$ cause problems in conjunction with the axioms of Boolean algebra and the exchange law, we should drop these. On the other hand, the converse hypotheses, i.e., $\{p \wedge q \leq p \cdot q \ : \ p, q \in \mathcal{T}\}$, are not involved in the derivation. Intuitively, these still make sense for concurrency: the behaviour of asserting p and then q includes the behaviour of asserting that p and q hold at the same time, in the special case that nothing happens between p and q. We refer to this set of hypotheses as the *contraction law*. This leaves us with the following:

$$\mathsf{ckat}' = \mathsf{exch} \cup \mathsf{bool} \cup \mathsf{choice} \cup \mathsf{contr} \cup \mathsf{unit} \quad \text{where} \quad \mathsf{contr} = \{p \wedge q \leq p \cdot q \ : \ p, q \in \mathcal{T}_\mathsf{B}\}$$

To see that ckat' does not permit the same strange derivation as ckat, we observe the following.

Fact 6.8. *There exist $p \in \mathcal{T}_\mathsf{B}$ and $e \in \mathcal{T}$ such that $p \cdot e \cdot \overline{p} \not\equiv^{\mathsf{ckat}'} 0$.*

Proof. Take $p = o \in \Omega$ and $e = \mathsf{a} \in \Sigma$, and let L be the set of all pomsets such that at least one event is labelled with a $q \in \mathcal{T}_\mathsf{B}$ such that $q \equiv_\mathsf{B} \bot$. First, note that if $e \leq f \in \mathsf{ckat}'$ such that $[\![f]\!] \subseteq L$, then $[\![e]\!] \subseteq L$. Since $[\![0]\!] \subseteq L$, a simple inductive argument shows that $[\![0]\!]^{\mathsf{ckat}'} \subseteq L$ as well. On the other hand, $o \cdot \mathsf{a} \cdot \overline{o} \in [\![p \cdot e \cdot \overline{p}]\!]$, and since $o, \overline{o} \not\equiv_\mathsf{B} \bot$, we have that $[\![p \cdot e \cdot \overline{p}]\!]^{\mathsf{ckat}'} \not\subseteq L$. This implies that $[\![p \cdot e \cdot \overline{p}]\!]^{\mathsf{ckat}'} \neq [\![0]\!]^{\mathsf{ckat}'}$, and hence $p \cdot e \cdot \overline{p} \not\equiv^{\mathsf{ckat}'} 0$ by Theorem 4.14. □

We are not out of the woods just yet. To see this, note that kat enjoys a useful property: any program whose behaviour is included in that of an assertion is equivalent to an assertion — that is, assertions can be used to describe all assertion-like behaviour. Furthermore, all equivalences of assertions follow from the laws of Boolean algebra — in other words, equivalence of assertions is mediated by Boolean algebra *exclusively*. Formally, we describe this property as follows.

Definition 6.9 (Conservative)**.** Let H be a set of hypotheses. We say that H is *conservative* if

(i) if $e \in \mathcal{T}_{\mathsf{GSR}}$ and $p \in \mathcal{T}_{\mathsf{B}}$ with $e \leqq^{H} p$, then there exists a $q \in \mathcal{T}_{\mathsf{B}}$ such that $e \equiv^{H} q$, and

(ii) if $p, q \in \mathcal{T}_{\mathsf{B}}$ such that $p \equiv^{H} q$, then $p \equiv_{\mathsf{B}} q$.

It is not entirely unreasonable, or indeed undesirable, that ckat' is conservative. Unfortunately, this means that the old (erroneous) set of hypotheses rears its head once more.

Fact 6.10. *If* ckat' *is conservative, then* ckat' *implies* ckat.

Proof. It suffices to verify that, for $p, q \in \mathcal{T}_{\mathsf{B}}$ we have $p \cdot q \leqq^{\mathsf{ckat}'} p \wedge q$. To this end, note that $p \cdot q \leqq^{\mathsf{ckat}'} p \cdot \top \leqq^{\mathsf{ckat}'} p \cdot 1 \equiv^{\mathsf{ckat}'} p$, which means that we obtain $p' \in \mathcal{T}_{\mathsf{B}}$ such that $p \cdot q \equiv^{\mathsf{ckat}'} p'$ by the first property of conservative hypotheses. Since $p' \equiv p \cdot q \leqq^{\mathsf{ckat}'} p \cdot 1 \equiv p$, we find that $p' \leqq^{\mathsf{ckat}'} p$ by the second property of conservative hypotheses. Likewise, we obtain $q' \in \mathcal{T}_{\mathsf{B}}$ such that $p \cdot q \equiv^{\mathsf{ckat}'} q'$ and $q' \leqq_{\mathsf{B}} q$. Since $p' \equiv^{\mathsf{ckat}'} q'$ and hence $p' \equiv_{\mathsf{B}} q'$, we can derive that $p \cdot q \equiv^{\mathsf{ckat}'} p' \equiv_{\mathsf{B}} p' \wedge q' \leqq_{\mathsf{B}} p \wedge q$. □

Thus, if ckat' were conservative, then for $p \in \mathcal{T}_{\mathsf{B}}$ and $e \in \mathcal{T}$, we would have $p \cdot e \cdot \overline{p} \equiv^{\mathsf{ckat}'} 0$ again. This contradicts Fact 6.8, which means that ckat' cannot be conservative. To work around this, note that the derivation above hinges on the hypothesis $\top = 1$ to show $p \cdot q \leqq^{\mathsf{ckat}'} p$ and $p \cdot q \leqq^{\mathsf{ckat}'} q$. If we drop this hypothesis, we arrive at the axioms of *Concurrent Kleene algebra with Observations*:

$$\mathsf{ckao} = \mathsf{exch} \cup \mathsf{bool} \cup \mathsf{choice} \cup \mathsf{contr} \cup \{0 = \bot\}$$

Remark 6.11. The derivation in Fact 6.10 can be prevented if we just omit the hypothesis $\top \leq 1$, and leave $1 \leq \top$ in place. We drop the latter hypothesis as well, for the sake of simplicity: we do not gain a lot of additional reasoning power by stipulating that the behaviour of doing nothing is contained in the behaviour of the assertion that always succeeds, but doing so would complicate our analysis a great deal. We defer the extension of ckao with this hypothesis to future work.

Note that ckao has more hope of being conservative, because the second property is a consequence of the technical lemma below. We shall prove the other property at the end of this chapter. Before we do, we first argue that ckao is decidable and complete, by reducing it to the empty set.

Lemma 6.12. *Let* $p, q \in \mathcal{T}_{\mathsf{B}}$*; then* $q \leqq_{\mathsf{B}} p$ *if and only if* $q \in [\![p]\!]^{\mathsf{ckao}}$.

6.2 Discharging the Boolean structure

The first step in reducing ckao to the empty set of hypotheses is to reduce the hypotheses that describe the Boolean structure on assertions. We do this by means of reification (c.f. Section 4.2.1), replacing every assertion by an expression that uses a restricted form of assertion.

This restricted form essentially spells out the states of the machine that would validate the assertion. For instance, if $\Omega = \{o_1, o_2\}$, then the assertion o_1 would be valid if both o_1 and o_2 were true, or if o_1 were true and o_2 were false. Syntactically, this is represented by the expression $(o_1 \wedge o_2) \vee (o_1 \wedge \overline{o_2})$. Readers familiar with Boolean algebra may recognise the latter as a disjunction of propositional expressions known as *atoms* [BB70]. We formalise these as follows.

Definition 6.13 (Atoms)**.** Let $\Omega = \{o_1, \ldots, o_n\}$. An *atom* is an element of $\mathcal{T}_{\mathsf{B}}$ of the form $t_1 \wedge \cdots \wedge t_n$, where for $1 \leq i \leq n$, we have that $t_i = o_i$ or $t_i = \overline{o_i}$. We write At for the set of atoms.

Example 6.14. If Ω is as above, then $o_1 \wedge o_2$ and $o_1 \wedge \overline{o_2}$ are indeed atoms. However, o_1 is not an atom (it does not mention o_2), and neither is $o_1 \vee o_2$, because it contains a disjunction.

An atom asserts for each of the variables whether it is true or false; they correspond with the *guards* that appear in guarded languages [KS96]. Note that there are finitely many atoms. It is well-known that every propositional expression can be written uniquely (up to commutativity and associativity) as the disjunction of the atoms below it, as follows.

Theorem 6.15 [BB70, Chapter 5.6]**.** *For every $p \in \mathcal{T}_{\mathsf{B}}$, it holds that*

$$p \equiv_{\mathsf{B}} \bigvee \{\alpha \in \mathsf{At} \;:\; \alpha \leqq_{\mathsf{B}} p\}$$

Furthermore, the expression on the right-hand side of the above equivalence is computable.

Since every propositional expression is equivalent to a disjunction of atoms, and choice identifies disjunction of propositions with their sum, we can rewrite every propositional expression to a sum of atoms. This prompts our candidate reification $r : \mathcal{T}_{\mathsf{GSR}} \to \mathcal{T}(\Gamma)$, where $\Gamma = \Sigma \cup \mathsf{At}$ and

$$r(a) = \begin{cases} \mathtt{a} & a = \mathtt{a} \in \Sigma \\ \sum \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p\} & a = p \in \mathcal{T}_{\mathsf{B}} \end{cases} \qquad r(e + f) = r(e) + r(f) \qquad \cdots$$

The question then arises: which hypotheses do we need to reason about expressions in $\mathcal{T}(\Gamma)$? First, note that since atoms do not contain any disjunction, and since $\bot$ is not an atom, we should be able to do away with the hypotheses in choice, as well as $0 = \bot$. As a matter of fact, every propositional expression has been normalised, so we also should not have to reason about equivalence between propositional expressions; this means that we should be able to reason without bool. Lastly, we need a version of the contraction law that is exclusive to atoms, since those are the only propositional expressions left. This leaves us with the following reduced set of hypotheses:

$$\mathsf{ckao}' = \mathsf{exch} \cup \mathsf{contr}' \quad \text{where} \quad \mathsf{contr}' = \{\alpha \leq \alpha \cdot \alpha \;:\; \alpha \in \mathsf{At}\}$$

We can now verify that r is a reification, and in fact a reduction, from ckao to ckao'.

Lemma 6.16. *The function r is a reduction from ckao to ckao'.*

Proof. By Lemma 4.32, it suffices to verify that ckao implies ckao', and that r is a reification from ckao to ckao'. First, note that ckao trivially implies exch. It remains to show that ckao implies contr'. To this end, let $\alpha \in \mathsf{At}$, and note that $\alpha \equiv_{\mathsf{B}} \alpha \wedge \alpha \leqq^{\mathsf{contr}} \alpha \cdot \alpha$, and therefore $\alpha \leqq^{\mathsf{ckao}} \alpha \cdot \alpha$.

The first requirement on reification holds by construction of r and the hypotheses from choice. Concretely, we have for $\mathsf{a} \in \Sigma$ that $r(\mathsf{a}) = \mathsf{a}$, and for $p \in \mathcal{T}_{\mathsf{B}}$ we can derive using Theorem 6.15 that

$$r(p) = \sum \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p\} \equiv^{\mathsf{choice}} \bigvee \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p\} \equiv_{\mathsf{B}} p$$

For the second requirement, let $e \leq f \in \mathsf{ckao}$; we should show that $r(e) \leqq^{\mathsf{ckao}'} r(f)$. To this end, we consider the different parts that make up ckao.

- If $e \leq f \in \mathsf{exch}$, then there exist $g_1, g_2, h_1, h_2 \in \mathcal{T}_{\mathsf{GSR}}$ such that $e = (g_1 \parallel h_1) \cdot (g_2 \parallel h_2)$ and $f = g_1 \cdot g_2 \parallel h_1 \cdot h_2$. In that case, $r(e) = (r(g_1) \parallel r(h_1)) \cdot (r(g_2) \parallel r(h_2))$ and $r(f) = r(g_1) \cdot r(g_2) \parallel r(h_1) \cdot r(h_2)$. Thus $r(e) \leq r(f) \in \mathsf{exch} \subseteq \mathsf{ckao}'$, and hence $r(e) \leqq^{\mathsf{ckao}'} r(f)$.
- If $e \leq f \in \mathsf{bool}$, then $e = p \in \mathcal{T}_{\mathsf{B}}$ and $f = q \in \mathcal{T}_{\mathsf{B}}$ such that $p \leqq_{\mathsf{B}} q$. Let $\alpha \in \mathsf{At}$ such that $\alpha \leqq_{\mathsf{B}} p$; it then follows that $\alpha \leqq_{\mathsf{B}} q$. Hence, every term in the sum that makes up $r(e)$ appears in $r(f)$, which means that $r(e) \leqq r(f)$.
- Let $e \leq f \in \mathsf{choice}$. There are two subcases. First, if $p, q \in \mathcal{T}_{\mathsf{B}}$ such that $e = p + q$ and $f = p \vee q$, then we can derive as follows

  $$r(e) = r(p) + r(q) = \sum \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p\} + \sum \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p\} \equiv \sum \{\alpha \;:\; \alpha \leqq_{\mathsf{B}} p \vee q\} = r(f)$$

  This also shows that if $e = p \vee q$ and $f = p + q$, then $r(e) \leqq r(f)$.

- Let $e \leq f \in \mathsf{contr}$, i.e., there exist $p, q \in \mathcal{T}_{\mathsf{B}}$ such that $e = p \wedge q$ and $f = p \cdot q$. We then derive

$$\begin{aligned}
r(e) &= \sum \{\alpha \ : \ \alpha \leqq_{\mathsf{B}} p \wedge q\} \\
&\leqq^{\mathsf{contr}'} \sum \{\alpha \cdot \alpha \ : \ \alpha \leqq_{\mathsf{B}} p \wedge q\} \\
&\leqq \sum \{\alpha \cdot \beta \ : \ \alpha \leqq_{\mathsf{B}} p, \beta \leqq_{\mathsf{B}} q\} \\
&\equiv \Big(\sum \{\alpha \ : \ \alpha \leqq_{\mathsf{B}} p\}\Big) \cdot \Big(\sum \{\alpha \ : \ \alpha \leqq_{\mathsf{B}} q\}\Big) && \text{(distributivity)} \\
&= r(p) \cdot r(q) = r(f)
\end{aligned}$$

- Finally, if $e = \bot$ and $f = 0$, then $r(e)$ is the empty sum, which by definition is 0. We can then conclude that $r(e) = 0 = r(f)$. □

6.3 Reducing the contraction law

The next step to show that ckao reduces to the empty set of hypotheses is to argue that the residual set of hypotheses ckao′ reduces to the empty set of hypotheses, on sr-expressions over Γ. Since ckao′ consists of exch and contr′, we can use what we learned in Section 5.2: because contr′ is left-simple, it suffices to find a strong reduction from contr′ to the empty set of hypotheses.

In this section, we show how to obtain such a strong reduction. The first step is to observe that contr′ is grounded. By Lemma 4.41, it then suffices to find a sequential reduction of contr′ to the empty set of hypotheses, and hence we can focus on rational expressions over $\Gamma = \Sigma \cup \mathsf{At}$.

Remark 6.17. The sequential reduction that we are about to demonstrate can also be obtained by means of the more general framework from [KM14]. We include our own construction because it saves us from having to import the required background on finitely presented monoids; the ideas behind why this construction works, however, can be traced back to op. cit.

Recall that a sequential reduction from contr′ to the empty set of hypotheses should show how to compute, for every $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$, an expression $\hat{e} \in \mathcal{T}_{\mathsf{R}}(\Gamma)$ such that both of the following hold:

$$\hat{e} \leqq_{\mathsf{R}}^{\mathsf{contr}'} e \leqq_{\mathsf{R}} \hat{e} \qquad\qquad [\![e]\!]^{\langle \mathsf{contr}' \rangle} = [\![\hat{e}]\!]$$

Example 6.18. If $e = (\alpha \cdot \alpha)^*$ for $\alpha \in \mathsf{At}$, then $\hat{e} = \alpha^*$ satisfies the constraints above. First, since $\alpha \leq \alpha \cdot \alpha \in \mathsf{contr}'$, we have that $\alpha^* \leqq_{\mathsf{R}}^{\mathsf{contr}'} (\alpha \cdot \alpha)^*$ by monotonicity. Second, since $\alpha \cdot \alpha \cdot \alpha^* + 1 \leqq_{\mathsf{R}} \alpha \cdot \alpha^* + 1 \equiv_{\mathsf{R}} \alpha^*$, we have that $(\alpha \cdot \alpha)^* \leqq_{\mathsf{R}} \alpha^*$ by the fixpoint rule. This also shows that $[\![\hat{e}]\!] \subseteq [\![e]\!]^{\langle \mathsf{contr}' \rangle}$. For the converse, note that any word in $[\![e]\!]$ contains α exclusively, and that contr′-closure preserves this; hence $[\![e]\!]^{\langle \mathsf{contr}' \rangle} \subseteq [\![\hat{e}]\!]$.

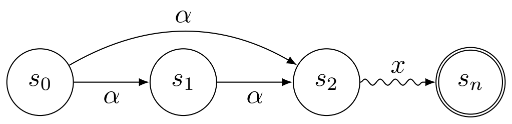


Figure 6.1: A shortcut in a non-deterministic automaton.

To get an idea for how the second requirement can be accomplished, consider the sequential contr'-closure. Here, $L^{\langle \mathsf{contr}' \rangle}$ for $L \subseteq \Gamma^*$ is the smallest set containing L, such that when $w \cdot \alpha \cdot \alpha \cdot x \in L^{\langle \mathsf{contr}' \rangle}$ with $w, x \in \Gamma^*$ and $\alpha \in \mathsf{At}$, also $w \cdot \alpha \cdot x \in L^{\langle \mathsf{contr}' \rangle}$. In other words, adjacent atoms can be merged, and every word in the contr'-closure of L should be obtainable this way. As a matter of fact, we can set up this series of merges such that the last merge is furthest to the left of the word. This leads us to the following characterisation of sequential closure w.r.t. contr'.

Lemma 6.19. *Let $L \subseteq \Gamma^*$, and define $\lessdot$ as the smallest preorder on Γ^* satisfying the rules*

$$\frac{\mathfrak{a} \in \Gamma \qquad w \trianglelefteq x}{\mathfrak{a} \cdot w \trianglelefteq \mathfrak{a} \cdot x} \qquad\qquad \frac{\alpha \in \mathsf{At} \qquad w \trianglelefteq x}{\alpha \cdot w \trianglelefteq \alpha \cdot \alpha \cdot x}\,.$$

Now $w \in L^{\langle \mathsf{contr}' \rangle}$ if and only if there exists an $x \in L$ such that $w \trianglelefteq x$.

In other words, to construct $\hat{e}$, we need to ensure that $[\![\hat{e}]\!]$ is the smallest $\trianglelefteq$-closed language that contains $[\![e]\!]$. To accomplish this, imagine an automaton accepting $[\![\hat{e}]\!]$: if a state s_0 accepts the word $\alpha \cdot \alpha \cdot x$, then it should also accept $\alpha \cdot x$. The path from s_0 to an accepting state for $\alpha \cdot \alpha \cdot x$ includes two transitions labelled α, the last of which leads to a state s_2. To make s_0 accept $\alpha \cdot x$, we could also include an α-transition from s_0 to s_2, bypassing the intermediate state (c.f. Figure 6.1).

We will use this intuition to guide our construction of $\hat{e}$. The general setup will be to construct a rational system represented by a non-deterministic automaton that implements the behaviour expected of $\hat{e}$, with the aforementioned shortcut transitions. The states in this automaton will be expressions; the idea is that a state $e \in \mathcal{T}_\mathsf{R}(\Gamma)$ represents the ($\trianglelefteq$-closed) language accepted by that expression, and a transition from e to e' labelled $\mathfrak{a} \in \Gamma$ means that, after reading $\mathfrak{a}$, we still need to read a word described by e'. The solution of this rational system will then be used to build $\hat{e}$.

We start with transitions exiting a state labelled with letters from Σ. These transitions are built inductively on the structure of the expression represented by the state, and exactly match the well-known partial derivatives of rational expressions proposed by Antimirov [Ant96].

Convention 6.20. If $S \subseteq \mathcal{T}_\mathsf{R}(\Gamma)$ and $e \in \mathcal{T}_\mathsf{R}(\Gamma)$, we abbreviate $\{f \cdot e \ : \ f \in S\}$ by writing $S \mathbin{⨾} e$.

Definition 6.21 (Σ-transitions)**.** We define $\delta : \mathcal{T}_{\mathsf{R}}(\Gamma) \times \Sigma \to 2^{\mathcal{T}_{\mathsf{R}}(\Gamma)}$ inductively

$$\begin{aligned} \delta(0, \mathtt{a}) &= \delta(1, \mathtt{a}) = \emptyset & \delta(e + f, \mathtt{a}) &= \delta(e, \mathtt{a}) \cup \delta(f, \mathtt{a}) \\ \delta(\mathtt{a}, \mathtt{b}) &= \{1 \ : \ \mathtt{a} = \mathtt{b}\} & \delta(e \cdot f, \mathtt{a}) &= \delta(e, \mathtt{a}) \fatsemi f \cup \Delta(e, f, \mathtt{a}) \\ \delta(\alpha, \mathtt{a}) &= \emptyset & \delta(e^*, \mathtt{a}) &= \delta(e, \mathtt{a}) \fatsemi e^* \end{aligned}$$

where $\Delta(e, f, \mathtt{a}) = \delta(f, \mathtt{a})$ when $e \in \mathcal{F}_{\mathsf{R}}$, and $\emptyset$ otherwise.

The interpretation of δ is that if $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$ and $\mathtt{a} \in \Sigma$, with $e' \in \delta(e, \mathtt{a})$, then e has a transition labelled $\mathtt{a}$ to e'. For instance, if $e = \mathtt{a}^*$, then $1 \in \delta(\mathtt{a}, \mathtt{a})$, and therefore $1 \cdot \mathtt{a}^* \in \delta(\mathtt{a}^*, \mathtt{a})$. Furthermore, since $1 \in \mathcal{F}_{\mathsf{R}}$ we have that $1 \cdot \mathtt{a}^* \in \Delta(1, \mathtt{a}^*, \mathtt{a})$, which means that $1 \cdot \mathtt{a}^* \in \delta(1 \cdot \mathtt{a}^*, \mathtt{a})$.

Before we give a more extensive example of the automata built using δ, we also need to define the transitions labelled by atoms. These are built in a manner very similar to the ones above; the only difference is that we need to add the shortcut edges.

Definition 6.22 (At-transitions)**.** We define $\zeta : \mathcal{T}_{\mathsf{R}}(\Gamma) \times \mathsf{At} \to 2^{\mathcal{T}_{\mathsf{R}}(\Gamma)}$ inductively

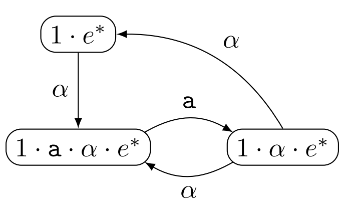
$$\begin{aligned} \zeta(0, \alpha) &= \delta(1, \alpha) = \emptyset & \zeta(e + f, \alpha) &= \zeta(e, \alpha) \cup \zeta(f, \alpha) \\ \zeta(\mathtt{a}, \alpha) &= \emptyset & \zeta(e \cdot f, \alpha) &= \zeta(e, \alpha) \fatsemi f \cup Z(e, f, \alpha) \\ \zeta(\alpha, \beta) &= \{1 \ : \ \alpha = \beta\} & \zeta(e^*, \alpha) &= \zeta(e, \alpha) \fatsemi e^* \end{aligned}$$

where $Z(e, f, \alpha) = \zeta(f, \alpha)$ when $e \in \mathcal{F}_{\mathsf{R}}$ or $e' \in \mathcal{F}_{\mathsf{R}}$ for some $e' \in \zeta(e, \alpha)$, and $\emptyset$ otherwise.

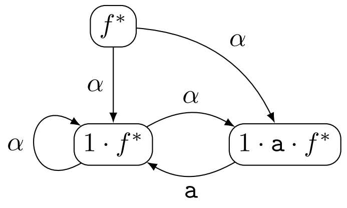
Because the transitions are built by induction on the expression, shortcut edges carry over quite nicely. For instance, the transitions that exit an expression like $e + f$ are built from the transitions exiting e and f; since these already have shortcut edges, we do not need to add them. We do need to take care for expressions of the form $e \cdot f$. Here, an atom-labelled edge exiting e may lead to an accepting state, in which case we should be able to skip ahead and perform an α-transition in f.

Example 6.23. To get an idea for what the transition structure generated by δ and ζ might look like, suppose $e = \alpha \cdot \mathtt{a} \cdot \alpha$. In that case, $1 \cdot \mathtt{a} \cdot \alpha \in \zeta(e, \alpha)$, and therefore $1 \cdot \mathtt{a} \cdot \alpha \cdot e^* \in \zeta(1 \cdot e^*, \alpha)$. Similarly, $1 \cdot \alpha \cdot e^* \in \delta(1 \cdot \mathtt{a} \cdot \alpha \cdot e^*, \mathtt{a})$, and $1 \cdot e^* \in \zeta(1 \cdot \alpha \cdot e^*)$. However, since $1 \cdot e^* \in \mathcal{F}_{\mathsf{R}}$, we have $1 \cdot \mathtt{a} \cdot \alpha \in \zeta(e^*, \alpha) = Z(1 \cdot \alpha, e^*) \subseteq \zeta(1 \cdot \alpha \cdot e^*, \alpha)$. This shortcuts the α-transitions that allow us to move from $1 \cdot \alpha \cdot e^*$ to $1 \cdot \mathtt{a} \cdot \alpha \cdot e^*$ by way of $1 \cdot e^*$. The transitions are depicted in Figure 6.2a.

Example 6.24. If f^* has two successive α-transitions, then the first transition may originate from f and lead to an accepting state, in which case the second transition is already available from f^*.


$1 \cdot e^*$
α
α
$\mathtt{a}$
$1 \cdot \mathtt{a} \cdot \alpha \cdot e^*$
$1 \cdot \alpha \cdot e^*$
α

(a) Transitions from $1 \cdot e^*$, where $e = \alpha \cdot \mathtt{a} \cdot \alpha$.



(b) Transitions from f^*, where $f = \alpha + \alpha \cdot \mathtt{a}$.

Figure 6.2: Schematic depictions of transitions generated by the derivative functions δ and ζ. If a node is labelled by an expression $g \in \mathcal{T}_\mathsf{R}(\Gamma)$ and has an edge labelled by $\mathtt{a} \in \Sigma$ to a node $g' \in \mathcal{T}_\mathsf{R}(\Gamma)$, this means that $g' \in \delta(g, \mathtt{a})$. Similarly, if the edge is labelled by $\alpha \in \mathsf{At}$, then $g' \in \zeta(g, \alpha)$.

For instance, if $f = \alpha + \alpha \cdot \mathtt{a}$, then $1 \in \zeta(f, \alpha)$, and hence $1 \cdot f^* \in \zeta(f^*, \alpha)$. Next, since $1 \cdot \mathtt{a} \in \zeta(f, \alpha)$, we have $1 \cdot \mathtt{a} \cdot f^* \in \zeta(1 \cdot f^*, \alpha)$. Thus, two α-transitions get us from f^* to $1 \cdot \mathtt{a} \cdot f^*$. However, since $1 \cdot \mathtt{a} \cdot f^* \in \zeta(f^*, \alpha)$, a single α-transition from f^* also exists. This is depicted in Figure 6.2b.

In the above examples, we see that the shortcut edges naturally appear as a result of the definition of ζ. To formally see that this is always the case, we record the following lemma.

Lemma 6.25. *Let $e \in \mathcal{T}_\mathsf{R}(\Gamma)$, $\alpha \in \mathsf{At}$. If $e' \in \zeta(e, \alpha)$, then $\zeta(e', \alpha) \subseteq \zeta(e, \alpha)$.*

Now, to build a rational system where states are expressions using the transition structures induced by δ and ζ, we need to make sure that the set of states in that rational system is closed under the transitions. To this end, we introduce the notion of the *reach* of an expression.

Definition 6.26 (Reach)**.** For $e \in \mathcal{T}_\mathsf{R}(\Gamma)$, we define the *reach* of e, denoted $\rho(e)$, inductively:

$$\rho(0) = \emptyset \qquad \rho(\mathtt{a}) = \{1, \mathtt{a}\} \qquad \rho(e + f) = \rho(e) \cup \rho(f) \qquad \rho(e^*) = \{1\} \cup \rho(e) \fatsemi e^*$$
$$\rho(1) = \{1\} \qquad \rho(\alpha) = \{1, \alpha\} \qquad \rho(e \cdot f) = \rho(e) \fatsemi f \cup \rho(f)$$

Example 6.27. Let $e = \alpha \cdot \alpha$. We can then calculate $\rho(1 \cdot e^*)$. To do this, we need to calculate $\rho(e^*)$, and for this we need $\rho(e)$. The latter is given by $\rho(\alpha \cdot \alpha) = \rho(\alpha) \fatsemi \alpha \cup \rho(\alpha) = \{1 \cdot \alpha, \alpha \cdot \alpha, 1, \alpha\}$. Thus, $\rho(e^*) = \{1 \cdot \alpha \cdot e^*, \alpha \cdot \alpha \cdot e^*, 1 \cdot e^*, \alpha \cdot e^*, 1\}$, which happens to be the same as $\rho(1 \cdot e^*)$.

It should be clear that, for a given $e \in \mathcal{T}_\mathsf{R}(\Gamma)$, the set $\rho(e)$ is finite; after all, every step to build $\rho(e)$ from its subexpressions combines finite sets in a finitary manner. We should also check that $\rho(e)$ is closed under following transitions stipulated by δ and ζ. This turns out to be the case.

Lemma 6.28. *Let $e \in \mathcal{T}_\mathsf{R}(\Gamma)$, and $e' \in \rho(e)$. The following hold.*

(i) For every $\mathtt{a} \in \Sigma$, it holds that $\delta(e', \mathtt{a}) \subseteq \rho(e)$.

(ii) For every $\alpha \in \mathsf{At}$, it holds that $\zeta(e', \alpha) \subseteq \rho(e)$.

Now, $\rho(e)$ does not, in general, contain e itself. On the surface, this could be fixed by defining $\rho(e)$ to contain e. For our purposes, however, it is more convenient to note that $\rho(e)$ contains enough information to reconstruct e. The subset of $\rho(e)$ necessary to do this is built as follows.

Definition 6.29 (Initial factors)**.** For $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$, we define $\iota : \mathcal{T}_{\mathsf{R}}(\Gamma) \to 2^{\mathcal{T}_{\mathsf{R}}(\Gamma)}$ inductively:

$$\begin{array}{llll} \iota(0) = \emptyset & \iota(\mathtt{a}) = \{\mathtt{a}\} & \iota(e + f) = \iota(e) \cup \iota(f) & \iota(e^*) = \{1\} \cup \iota(e) \fatsemi e^* \\ \iota(1) = \{1\} & \iota(\alpha) = \{\alpha\} & \iota(e \cdot f) = \iota(e) \fatsemi f & \end{array}$$

Example 6.30. To compute $\iota(1 \cdot e^*)$ where $e = \alpha \cdot \alpha$, we should calculate $\iota(e^*)$. To this end, we first note that $\iota(e) = \{\alpha \cdot \alpha\}$, and hence $\iota(e^*) = \{1, \alpha \cdot \alpha \cdot e^*\}$. This happens to be $\iota(1 \cdot e^*)$, too. We can now add the expressions in $\iota(1 \cdot e^*)$ to find that $1 + \alpha \cdot \alpha \cdot e^* \equiv_{\mathsf{R}} e^* \equiv_{\mathsf{R}} 1 \cdot e^*$.

It should be clear that $\iota(e)$ is contained in $\rho(e)$ by construction. In the above, we also saw that adding the contents of $\iota(e)$ together yielded an expression equivalent to e. This works in general.

Lemma 6.31. *Let $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$. Then $e \equiv_{\mathsf{R}} \sum \iota(e)$.*

We now have everything in place to define a rational system on $\rho(e)$, with transitions brokered by the functions δ and ζ. Because we want this rational system to contain the original behaviour of e, each state can also terminate if the expression it represents can accept the empty word.

Definition 6.32. Let $e \in \mathcal{T}$. We define the rational system $\mathcal{S}_e = \langle M_e, b_e \rangle$ on $\rho(e)$ by setting

$$M_e(e', e'') = \sum_{e'' \in \delta(e', \mathtt{a})} \mathtt{a} + \sum_{e'' \in \zeta(e', \alpha)} \alpha \qquad\qquad b_e(e') = [e' \in \mathcal{F}_{\mathsf{R}}]$$

We denote the least solution to this rational system (c.f. Theorem 4.43) by s_e. Furthermore, we write $\hat{e}$ for $\sum \{s_e(e') \ : \ e' \in \iota(e)\}$, i.e., the sum of least solutions to initial factors of e.

Example 6.33. If $e = \alpha \cdot \alpha$, then the rational system for e^* has five states, as calculated in Example 6.27. The constraints on a solution $s : \rho(e^*) \to \mathcal{T}_{\mathsf{R}}(\Gamma)$ to this rational system are

$$\begin{array}{cccccccc} & & \alpha \cdot s(1 \cdot \alpha \cdot e^*) & + & \alpha \cdot s(1 \cdot e^*) & \leqq_{\mathsf{R}} & s(1 \cdot \alpha \cdot e^*) \\ & & \alpha \cdot s(1 \cdot \alpha \cdot e^*) & + & \alpha \cdot s(1 \cdot e^*) & \leqq_{\mathsf{R}} & s(\alpha \cdot \alpha \cdot e^*) \\ 1 & + & \alpha \cdot s(1 \cdot \alpha \cdot e^*) & + & \alpha \cdot s(1 \cdot e^*) & \leqq_{\mathsf{R}} & s(1 \cdot e^*) \\ & & \alpha \cdot s(1 \cdot \alpha \cdot e^*) & + & \alpha \cdot s(1 \cdot e^*) & \leqq_{\mathsf{R}} & s(\alpha \cdot e^*) \\ 1 & & & & & \leqq_{\mathsf{R}} & s(1) \end{array}$$

To prove that the solution to $\mathcal{S}_e$ indeed gives us the desired expression, we need to analyse how the derivatives relate to containment up to contr'. In particular, we need to show that if e can take a transition labelled $\mathfrak{a} \in \Gamma$ to e', then the behaviour of $\mathfrak{a} \cdot e'$ is contained in the behaviour of e.

Lemma 6.34. *Let $e \in \mathcal{T}_\mathsf{R}(\Gamma)$. The following hold.*

(i) For all $\mathtt{a} \in \Sigma$ and $e' \in \delta(e, \mathtt{a})$, we have $\mathtt{a} \cdot e' \leqq_\mathsf{R}^{\mathsf{contr}'} e$.

(ii) For all $\alpha \in \mathsf{At}$ and $e' \in \zeta(e, \alpha)$, we have $\alpha \cdot e' \leqq_\mathsf{R}^{\mathsf{contr}'} e$.

Another thing that is useful is that the solution to an expression in $\mathcal{S}_e$ is below that expression.

Lemma 6.35. *If $e \in \mathcal{T}_\mathsf{R}(\Gamma)$ and $e' \in \rho(e)$, then $e' \leqq_\mathsf{R} s_e(e')$.*

With this analysis of $\mathcal{S}_e$, we are ready to show that it gives us the desired reduction.

Lemma 6.36. *The hypotheses in contr' sequentially reduce to the empty set.*

Proof. We start by proving that for $e' \in \rho(e)$ we have $s_e(e') \leqq_\mathsf{R}^{\mathsf{contr}'} e' \leqq_\mathsf{R} s_e(e')$ as well as $[\![e']\!]^{\langle \mathsf{contr}' \rangle} = [\![s_e(e')]\!]$. First, we note we already have $e' \leqq_\mathsf{R} s_e(e')$, by Lemma 6.35.

To show that $s_e(e') \leqq_\mathsf{R}^{\mathsf{contr}'} e'$, let $s : \rho(e) \to \mathcal{T}_\mathsf{R}(\Gamma)$ be given by $s(e') = e$. By Theorem 4.43 it suffices to show that s is a $\langle \equiv_\mathsf{R}^{\mathsf{contr}'}, 1\rangle$-solution to $\mathcal{S}_e$: since s_e is the *least* $\langle \equiv_\mathsf{R}^{\mathsf{contr}'}, 1\rangle$-solution, the claim then follows. To this end, first note that $b_e(e') = [e' \in \mathcal{F}_\mathsf{R}] \leqq_\mathsf{R} e'$. Furthermore, if $e'' \in \delta(e', \mathtt{a})$, then $\mathtt{a} \cdot e'' \leqq_\mathsf{R}^{\mathsf{contr}'} e'$, and if $e'' \in \zeta(e', \alpha)$, then $\alpha \cdot e'' \leqq_\mathsf{R}^{\mathsf{contr}'} e'$, by Lemma 6.34. Hence, $M_e(e', e'') \cdot s(e'') \leqq_\mathsf{R}^{\mathsf{contr}'} s(e')$. This makes s a $\langle \equiv_\mathsf{R}^{\mathsf{contr}'}, 1\rangle$-solution to $\mathcal{S}_e$, and thus $s_e(e') \leqq_\mathsf{R}^{\mathsf{contr}'} e'$.

Finally, we should show that $[\![e']\!]^{\langle \mathsf{contr}' \rangle} = [\![s_e(e')]\!]$. Note that the inclusion from right to left follows by soundness (Theorem 4.14) and the fact that $s_e(e') \leqq_\mathsf{R}^{\mathsf{contr}'} e'$. For the other inclusion, it suffices to prove that $[\![s_e(e')]\!]$ is closed under $\trianglelefteq$, by Lemma 6.19. Before we start, it is useful to note that by the fact that s_e is a solution to $\mathcal{S}_e$ and by soundness (Theorem 3.47), we have that

$$[\![s_e(e')]\!] = \{1 \ : \ e' \in \mathcal{F}_\mathsf{R}\} \cup \bigcup \{\alpha \cdot [\![s_e(e'')]\!] \ : \ e'' \in \zeta(e', \alpha)\} \cup \bigcup \{\mathtt{a} \cdot [\![s_e(e'')]\!] \ : \ e'' \in \delta(e', \mathtt{a})\}$$

Now, let $x \in [\![s_e(e')]\!]$ and let $w \trianglelefteq x$; we should show that $w \in [\![s_e(e')]\!]$ as well. The proof proceeds by induction on the construction of $\trianglelefteq$, in tandem for all $e' \in \rho(e)$. In the base, $w = x$, and hence $w \in [\![s_e(e')]\!]$ immediately. In the inductive step, there are two cases to consider.

- First, suppose $\mathfrak{a} \in \Gamma$ such that $w = \mathfrak{a} \cdot w'$ and $x = \mathfrak{a} \cdot x'$ with $w' \trianglelefteq x'$. If $\mathfrak{a} = \alpha \in \Gamma$, then since $\alpha \cdot x' \in [\![s_e(e')]\!]$ there exists an $e'' \in \zeta(e', \alpha)$ such that $x' \in [\![s_e(e'')]\!]$. By induction, it follows that $w' \in [\![s_e(e'')]\!]$, and therefore $w = \alpha \cdot w' \in [\![s_e(e')]\!]$. The case where $\mathfrak{a} = \mathtt{a} \in \Sigma$ is similar.

- Next, suppose $\alpha \in \mathsf{At}$ with $w = \alpha \cdot w'$ and $x = \alpha \cdot \alpha \cdot x'$ s.t. $w' \trianglelefteq x'$. Using the above, we find $e'' \in \zeta(e', \alpha)$ and $e''' \in \zeta(e'', \alpha)$ such that $x' \in [\![s_e(e''')]\!]$. By induction, we find $w' \in [\![s_e(e''')]\!]$. By Lemma 6.25, we have $e''' \in \zeta(e', \alpha)$. Hence, we conclude $w = \alpha \cdot w' \in [\![s_e(e')]\!]$.
- Lastly, if $w \trianglelefteq x$ because there exists a $y \in \Gamma^*$ such that $w \trianglelefteq y$ and $y \trianglelefteq x$, then by induction we find that $y \in [\![s_e(e')]\!]$, and applying the induction hypothesis once more we find $w \in [\![s_e(e')]\!]$.

To show that $[\![e]\!]^{\langle \mathsf{contr}' \rangle} = [\![\hat{e}]\!]$, we derive using Lemma 6.31 and the above that

$$\begin{aligned}
[\![e]\!]^{\langle \mathsf{contr}' \rangle} &= \left(\bigcup \{[\![e']\!] \;:\; e' \in \iota(e)\}\right)^{\langle \mathsf{contr}' \rangle} \\
&= \bigcup \left\{[\![e']\!]^{\langle \mathsf{contr}' \rangle} \;:\; e' \in \iota(e)\right\} \\
&= \bigcup \{[\![s_e(e')]\!] \;:\; e' \in \iota(e)\} = [\![\hat{e}]\!]
\end{aligned}$$

where we use that $(-)^{\langle \mathsf{contr}' \rangle}$ commutes with union, as a consequence of Lemma 6.19.

Finally, to show that $\hat{e} \leqq_{\mathsf{R}}^{\mathsf{contr}'} e \leqq \hat{e}$, we derive using Lemma 6.31 and the above that

$$\hat{e} = \sum \{s_e(e') \;:\; e' \in \iota(e)\} \leqq_{\mathsf{R}}^{\mathsf{contr}'} \sum \iota(e) \equiv_{\mathsf{R}} e \equiv_{\mathsf{R}} \sum \iota(e) \leqq_{\mathsf{R}} \sum \{s_e(e') \;:\; e' \in \iota(e)\} = \hat{e} \quad \square$$

It is now time to harvest the fruits of our labour and put together the results of the previous two sections. We start by composing the reductions to find a reduction for ckao.

Theorem 6.37. *The hypotheses in* ckao *reduce to the empty set.*

Proof. We saw that ckao reduces to ckao' in Lemma 6.16. Furthermore, contr' sequentially reduces to the empty set of hypotheses, by Lemma 6.36. Since contr' is grounded, this means that it strongly reduces to the empty set of hypotheses, by Lemma 4.41. Next, because contr' is left-simple, we have that $\mathsf{ckao}' = \mathsf{exch} \cup \mathsf{contr}'$ strongly reduces to the empty set of hypotheses, by Corollary 5.53. Composing these reductions, we find that ckao reduces to the empty set of hypotheses. □

As a direct consequence of the above and Lemma 4.27, we obtain the following.

Corollary 6.38. *Let* $e, f \in \mathcal{T}_{\mathsf{GSR}}$*. The following are true:*

(i) It holds that $e \equiv^{\mathsf{ckao}} f$ *if and only if* $[\![e]\!]^{\mathsf{ckao}} = [\![f]\!]^{\mathsf{ckao}}$.

(ii) It is decidable whether $[\![e]\!]^{\mathsf{ckao}} = [\![f]\!]^{\mathsf{ckao}}$ *(and hence, whether* $e \equiv^{\mathsf{ckao}} f$*).*

Finally, recall that in Section 6.1 we rejected ckat' because it was not *conservative*, but we left open the question of whether ckao was. Using completeness, we can settle this.

Theorem 6.39. *The set of hypotheses* ckao *is conservative.*

Proof. For the first property, suppose that $e \in \mathcal{T}_{\mathsf{GSR}}$ and $p \in \mathcal{T}_{\mathsf{B}}$ such that $e \leqq^{\mathsf{ckao}} p$. By Theorem 4.14, we have $[\![e]\!]^{\mathsf{ckao}} \subseteq [\![p]\!]^{\mathsf{ckao}}$. Choose $q = \bigvee \left\{\alpha \in \mathsf{At} \ : \ \alpha \in [\![e]\!]^{\mathsf{ckao}}\right\}$; note that this expression is well-defined, since At is finite. It remains to prove that $e \equiv^{\mathsf{ckao}} q$, which we do as follows.

- First, note that if $\alpha \in [\![e]\!]^{\mathsf{ckao}}$, then $[\![\alpha]\!]^{\mathsf{ckao}} \subseteq [\![e]\!]^{\mathsf{ckao}}$, and hence $\alpha \leqq^{\mathsf{ckao}} e$ by Corollary 6.38(i). We can then find that $q \equiv^{\mathsf{ckao}} \sum \left\{\alpha \in \mathsf{At} \ : \ \alpha \in [\![e]\!]^{\mathsf{ckao}}\right\} \leqq^{\mathsf{ckao}} e$.
- For the converse, note that by Corollary 6.38(i), it suffices to prove $[\![e]\!]^{\mathsf{ckao}} \subseteq [\![q]\!]^{\mathsf{ckao}}$. To this end, let $r \in [\![e]\!]^{\mathsf{ckao}}$. Then, if $\alpha \in \mathsf{At}$ such that $\alpha \leqq_{\mathsf{B}} r$, also $\alpha \in [\![e]\!]^{\mathsf{ckao}}$, and thus $\alpha \leqq_{\mathsf{B}} q$. By Theorem 6.15, $r \equiv_{\mathsf{B}} \bigvee \{\alpha \in \mathsf{At} \ : \ \alpha \leqq_{\mathsf{B}} r\} \leqq_{\mathsf{B}} q$, and therefore $r \in [\![q]\!]^{\mathsf{ckao}}$ by Lemma 6.12.

For the second property, let $p, q \in \mathcal{T}_{\mathsf{B}}$, and suppose that $p \equiv^{\mathsf{ckao}} q$. By Theorem 4.14, we have that $[\![p]\!]^{\mathsf{ckao}} = [\![q]\!]^{\mathsf{ckao}}$. But then, since $p \in [\![p]\!] \subseteq [\![p]\!]^{\mathsf{ckao}} = [\![q]\!]^{\mathsf{ckao}}$, we have that $p \leqq_{\mathsf{B}} q$ by Lemma 6.12. We find similarly that $q \leqq_{\mathsf{B}} p$, and hence $p \equiv_{\mathsf{B}} q$. □

Summary of this chapter Kleene algebra can be extended with a set of hypotheses to obtain Kleene algebra with tests, which adds the possibility of reasoning about control flow. We showed that, if one tries to extend bi-Kleene algebra along the same lines, the resulting system is unsuitable for reasoning about program equivalence. However, when the hypotheses that govern tests are weakened slightly, we obtain a new primitive for reasoning about control flow, called *concurrent Kleene algebra with observations*. Using the framework of reductions explored in the previous chapters, we obtained completeness and decidability results for this system.

6.A Proofs towards completeness

Lemma 6.12. *Let $p, q \in \mathcal{T}_{\mathsf{B}}$; then $q \leqq_{\mathsf{B}} p$ if and only if $q \in [\![p]\!]^{\mathsf{ckao}}$.*

Proof. The implication from left to right holds by definition of ckao-closure. For the other direction, we prove that if $q \in A \subseteq [\![p]\!]^{\mathsf{ckao}}$, then $A \subseteq \mathcal{T}_{\mathsf{B}}$ and $q \leqq_{\mathsf{B}} p$, by induction on $A \subseteq L^{\mathsf{ckao}}$. In the base, $A = [\![p]\!]$, which means that the claim holds immediately. For the inductive step, we obtain $e \leq f \in \mathsf{ckao}$ and $C \in \mathsf{PC}^{\mathsf{sp}}$ such that $C[[\![f]\!]] \subseteq [\![p]\!]^{\mathsf{ckao}}$ and $A = C[[\![e]\!]]$. By induction, we then have that $C[[\![f]\!]] \subseteq \mathcal{T}_{\mathsf{B}}$, which can mean one of two things.

- If $C \neq \square$, then $[\![f]\!] \subseteq \{1\}$. It is not hard to show that, since $e \leq f \in \mathsf{ckao}$, we have $[\![e]\!] \subseteq \{1, \bot\}$. This means that $q = \bot$, and therefore the claim holds immediately.
- If $C = \square$, then $[\![f]\!] \subseteq [\![p]\!]^{\mathsf{ckao}}$ and $p \in [\![e]\!]$. This gives us some cases to consider:

- If $e \leq f \in \mathsf{bool}$, then $e, f \in \mathcal{T}_\mathsf{B}$ such that $e \leqq_\mathsf{B} f$. Since $f \in [\![f]\!]^{\mathsf{ckao}}$, we obtain by induction that $f \leqq_\mathsf{B} p$. Since $q \in [\![e]\!]$, we have that $q = e \leqq_\mathsf{B} f \leqq_\mathsf{B} p$.
- If $e \leq f \in \mathsf{exch}$, then $q \in [\![e]\!]$ implies that $q \in [\![f]\!]$ as well; the claim follows by induction.
- If $e \leq f \in \mathsf{choice}$, then there are two subcases to consider.
  * If $e = r + s$ and $f = r \vee s$ for $r, s \in \mathcal{T}_\mathsf{B}$, then by induction we find $r \vee s \leqq_\mathsf{B} p$. Since $q \in [\![e]\!]$, we have $q = r$ or $q = s$; in either case, $q \leqq_\mathsf{B} r \vee s \leqq_\mathsf{B} p$.
  * If $e = r \vee s$ and $f = r + s$ for $r, s \in \mathcal{T}_\mathsf{B}$, then by induction we find that $r, s \leqq_\mathsf{B} p$. It then follows that $q = r \vee s \leqq_\mathsf{B} p$ as well.
- If $e \leq f \in \mathsf{contr}$, then $e = r \wedge s$ and $f = r \cdot s$ for $r, s \in \mathcal{T}_\mathsf{B}$. But this contradicts that $[\![f]\!] \subseteq \mathcal{T}_\mathsf{B}$, which we know by induction. We can therefore disregard this case.
- If $e \leq f \in \{0 = \bot\}$, then we can disregard the case where $e = 0$, for it contradicts that $q \in [\![e]\!]$. Thus we have that $e = \bot$. In that case $q = \bot$, and hence $q \leqq_\mathsf{B} p$. □

Lemma 6.19. *Let $L \subseteq \Gamma^*$, and define $\trianglelefteq$ as the smallest preorder on Γ^* satisfying the rules*

$$\frac{\mathfrak{a} \in \Gamma \qquad w \trianglelefteq x}{\mathfrak{a} \cdot w \trianglelefteq \mathfrak{a} \cdot x} \qquad\qquad \frac{\alpha \in \mathsf{At} \qquad w \trianglelefteq x}{\alpha \cdot w \trianglelefteq \alpha \cdot \alpha \cdot x}.$$

Now $w \in L^{\langle \mathsf{contr}' \rangle}$ if and only if there exists an $x \in L$ such that $w \trianglelefteq x$.

Proof. We first define $\blacktriangleleft$ as the smallest preorder on Γ^* satisfying

$$\frac{\alpha \in \mathsf{At} \qquad u, v \in \Gamma^*}{u \cdot \alpha \cdot v \blacktriangleleft u \cdot \alpha \cdot \alpha \cdot v}$$

We now claim that $\blacktriangleleft$ is the same as $\trianglelefteq$, which we prove as follows.

- Suppose that $w \blacktriangleleft x$; we proceed by induction on the construction of $\blacktriangleleft$. In the base, either $w = x$, in which case $w \trianglelefteq x$ immediately, or there exist $u, v \in \Gamma^*$ and $\alpha \in \mathsf{At}$ such that $w = u \cdot \alpha \cdot v$ and $x = u \cdot \alpha \cdot \alpha \cdot v$. Since $v \trianglelefteq v$, we have that $\alpha \cdot v \trianglelefteq \alpha \cdot \alpha \cdot v$. By induction on the length of u, we can then easily show that $w = u \cdot \alpha \cdot v \trianglelefteq u \cdot \alpha \cdot \alpha \cdot v = x$.
- Suppose that $w \trianglelefteq x$; we proceed by induction on the construction of $\trianglelefteq$. In the base, we have that $w = x$, in which case $w \blacktriangleleft x$. For the inductive step, there are two cases to consider.
  - If there exists an $\mathfrak{a} \in \Gamma$ such that $w = \mathfrak{a} \cdot w'$ and $x = \mathfrak{a} \cdot x'$ with $w' \trianglelefteq x'$, then by induction $w' \blacktriangleleft x'$. A simple inductive argument on the construction of $\blacktriangleleft$ then shows that $w = \mathfrak{a} \cdot w' \blacktriangleleft \mathfrak{a} \cdot x' = x$.

  - If there exists an $\alpha \in \mathsf{At}$ such that $w = \alpha \cdot w'$ and $x = \alpha \cdot \alpha \cdot x'$ with $w' \trianglelefteq x'$, then by induction $w' \blacktriangleleft x'$. By an argument similar to the previous case, we can show that $w = \alpha \cdot w' \blacktriangleleft \alpha \cdot x'$. Since $\alpha \cdot x' \blacktriangleleft \alpha \cdot \alpha \cdot x' = x$, it follows that $w \blacktriangleleft x$.

For the main claim, it suffices to show that $w \in L^{\langle \mathsf{contr}' \rangle}$ if and only if there exists an $x \in L$ with $w \blacktriangleleft x$. For the implication from left to right, suppose $w \in A \subseteq L^{\langle \mathsf{contr}' \rangle}$; we proceed by induction on $A \subseteq L^{\langle \mathsf{contr}' \rangle}$. In the base, we have that $A = L$, and hence $x = w$ suffices. For the inductive step, we obtain $\alpha \in \Gamma$ and $u, v \in \Gamma^*$ such that $A = u \cdot \{\alpha\} \cdot v$ and $u \cdot \{\alpha \cdot \alpha\} \cdot v \subseteq L^{\langle \mathsf{contr}' \rangle}$. By induction, we find $x \in L$ such that $u \cdot \alpha \cdot \alpha \cdot v \blacktriangleleft x$. Since $w = u \cdot \alpha \cdot v \blacktriangleleft u \cdot \alpha \cdot \alpha \cdot v \blacktriangleleft x$, the claim then follows.

For the other direction, we prove more generally that if $w \blacktriangleleft x \in L^{\langle \mathsf{contr}' \rangle}$, then $w \in L^{\langle \mathsf{contr}' \rangle}$. We proceed by induction on the construction of $w \blacktriangleleft x$. In the base, either $w = x$, in which case the claim holds immediately, or there exist $u, v \in \Gamma^*$ such that $w = u \cdot \alpha \cdot v$ and $x = u \cdot \alpha \cdot \alpha \cdot v$. In that case, since $u \cdot \{\alpha \cdot \alpha\} \cdot v \subseteq L$ and $\alpha \leq \alpha \cdot \alpha \in \mathsf{contr}'$, it follows that $w \in u \cdot \{\alpha\} \cdot v \subseteq L^{\langle \mathsf{contr}' \rangle}$. For the inductive step, we have that $w \blacktriangleleft x$ because there exists a $y \in \Gamma^*$ with $w \blacktriangleleft y$ and $y \blacktriangleleft x$. By induction, we find that $y \in L^{\langle \mathsf{contr}' \rangle}$, and hence again by induction we conclude that $w \in L^{\langle \mathsf{contr}' \rangle}$. □

Lemma 6.25. *Let $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$, $\alpha \in \mathsf{At}$. If $e' \in \zeta(e, \alpha)$, then $\zeta(e', \alpha) \subseteq \zeta(e, \alpha)$.*

Proof. We proceed by induction on e. In the base, where $e \in \{0, 1\} \cup \Gamma$, the claim holds, since necessarily $e' = 1$, and hence $\zeta(e', \alpha) = \emptyset$. For the inductive step, there are three cases.

- If $e = e_0 + e_1$, then $e' \in \zeta(e_0, \alpha)$ or $e' \in \zeta(e_1, \alpha)$; we assume the former without loss of generality. By induction, $\zeta(e', \alpha) \subseteq \zeta(e_0, \alpha)$; since $\zeta(e_0, \alpha) \subseteq \zeta(e, \alpha)$, the claim then follows.
- If $e = e_0 \cdot e_1$, we have three subcases to consider.
  - If $e' = e_0' \cdot e_1$ with $e_0' \in \zeta(e_0, \alpha)$, then by induction we know that $\zeta(e_0', \alpha) \subseteq \zeta(e_0, \alpha)$. If $e'' \in \zeta(e_0' \cdot e_1, \alpha)$, then we have two more subcases to consider.
    - If $e'' = e_0'' \cdot e_1$ for some $e_0'' \in \zeta(e_0', \alpha)$, then by induction we know that $e_0'' \in \zeta(e_0, \alpha)$, and therefore $e'' \in \zeta(e_0 \cdot e_1, \alpha)$.
    - If $e'' \in Z(e_0', e_1, \alpha) = \zeta(e_1, \alpha)$, then one of two cases applies. First, if $e_0' \in \mathcal{F}_{\mathsf{R}}$, then $e'' \in \zeta(e_1, \alpha) = Z(e_0, e_1, \alpha)$. Second, if there exists an $e_0'' \in \zeta(e_0', \alpha)$ such that $e_0'' \in \mathcal{F}_{\mathsf{R}}$, then $e_0'' \in \zeta(e_0, \alpha)$ by induction; hence, $e'' \in \zeta(e_1, \alpha) = Z(e_0, e_1, \alpha) \subseteq \zeta(e, \alpha)$.
  - If $e' \in Z(e_0', e_1, \alpha) = \zeta(e_1, \alpha)$ then by induction $e'' \in \zeta(e_1, \alpha)$. Furthermore, either $e_0' \in \mathcal{F}_{\mathsf{R}}$ or there exists $e_0'' \in \zeta(e_0', \alpha)$ with $e_0'' \in \mathcal{F}_{\mathsf{R}}$, in which case $e_0'' \in \zeta(e_0, \alpha)$ by induction. In either case, $Z(e_0, e_1, \alpha) \subseteq \zeta(e, \alpha)$, which completes this part of the proof.

- If $e = e_0^*$, then $e' = e_0' \cdot e_0^*$ for some $e_0' \in \zeta(e_0, \alpha)$. If $e'' \in \zeta(e', \alpha)$, then we have two subcases:
  - If $e'' = e_0'' \cdot e_0^*$ for some $e_0'' \in \zeta(e_0', \alpha)$, then by induction we know that $e_0'' \in \zeta(e_0, \alpha)$. It therefore follows that $e'' \in \zeta(e, \alpha)$.
  - If $e'' \in Z(e_0', e_0^*, \alpha) = \zeta(e_0^*, \alpha)$, then $e'' \in \zeta(e, \alpha)$ immediately. □

Lemma 6.28. *Let $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$, and $e' \in \rho(e)$. The following hold.*

(i) For every $\mathtt{a} \in \Sigma$, it holds that $\delta(e', \mathtt{a}) \subseteq \rho(e)$.

(ii) For every $\alpha \in \mathsf{At}$, it holds that $\zeta(e', \alpha) \subseteq \rho(e)$.

Proof. We prove the second claim; the proof of the first claim is similar. We start by proving that for $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$ we have $\zeta(e, \alpha) \subseteq \rho(e)$, by induction on e. In the base, where $e \in \{0, 1\} \cup \Gamma$, the claim holds, since $\zeta(e, \alpha) \subseteq \{1\} \subseteq \rho(e)$. For the inductive step, there are three cases.

- If $e = e_0 + e_1$, then by induction we have $\zeta(e_0, \alpha) \subseteq \rho(e_0)$ and $\zeta(e_1, \alpha) \subseteq \rho(e_1)$. Hence, we derive that $\zeta(e, \alpha) = \zeta(e_0, \alpha) \cup \zeta(e_1, \alpha) \subseteq \rho(e_0) \cup \rho(e_1) = \rho(e)$.
- If $e = e_0 \cdot e_1$, then by induction we have $\zeta(e_0, \alpha) \subseteq \rho(e_0)$ and $\zeta(e_1, \alpha) \subseteq \rho(e_1)$. Hence, we calculate that $\zeta(e, \alpha) = \zeta(e_0, \alpha) \fatsemi e_1 \cup Z(e_0, e_1, \alpha) \subseteq \rho(e_0) \fatsemi e_1 \cup \rho(e_1) = \rho(e)$.
- If $e = e_0^*$, then by induction we have $\zeta(e_0, \alpha) \subseteq \rho(e_0)$. Hence, we find that $\zeta(e, \alpha) = \zeta(e_0, \alpha) \fatsemi e_0^* \subseteq \rho(e_0) \fatsemi e_0^* \subseteq \rho(e)$.

For the main claim, it now suffices to show that if $e' \in \rho(e)$, then $\rho(e') \subseteq \rho(e)$. We proceed by induction on e. In the base, there are two cases. First, if $e \in \{0, 1\} \cup \Gamma$, then the claim holds immediately, because $e' \in \{0, 1\}$. For the inductive step, there are three cases to consider.

- If $e = e_0 + e_1$, assume w.l.o.g. $e' \in \rho(e_0)$. By induction, $\rho(e') \subseteq \rho(e_0) \subseteq \rho(e)$.
- If $e = e_0 \cdot e_1$ then there are two cases to consider.
  - If $e' = e_0' \cdot e_1$ where $e_0' \in \rho(e_0)$, then $\rho(e') = \rho(e_0') \fatsemi e_1 \cup \rho(e_1) \subseteq \rho(e_0) \fatsemi e_1 \cup \rho(e_1) = \rho(e)$ ..
  - If $e' \in \rho(e_1)$, then by induction we have $\rho(e') \subseteq \rho(e_1) \subseteq \rho(e)$.
- If $e = e_0^*$, then either $e' = 1$ or $e' = e_0' \cdot e_0^*$ for some $e_0' \in \rho(e_0)$. In the former case, $\rho(e') = \emptyset$. In the latter case, we find by induction that $\rho(e') = \rho(e_0') \fatsemi e_0^* \cup \rho(e_0^*) \subseteq \rho(e_0) \fatsemi e_0^* \cup \rho(e_0^*) = \rho(e)$. □

Lemma 6.31. *Let $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$. Then $e \equiv_{\mathsf{R}} \sum \iota(e)$.*

Proof. The proof proceeds by induction on e. In the base, where $e \in \{0, 1\} \cup \Gamma$, the claim holds trivially. For the inductive step, there are three cases to consider.

- If $e = e_0 + e_1$, then we calculate by induction that

$$e = e_0 + e_1 \equiv_{\mathsf{R}} \sum_{e'_0 \in \iota(e_0)} e'_0 + \sum_{e'_1 \in \iota(e_1)} e_1 \equiv_{\mathsf{R}} \sum_{e' \in \iota(e)} e' \ .$$

- If $e = e_0 \cdot e_1$, then we calculate by induction that

$$e = e_0 \cdot e_1 \equiv_{\mathsf{R}} \sum_{e'_0 \in \iota(e_0)} e'_0 \cdot e_1 \equiv_{\mathsf{R}} \sum_{e' \in \iota(e)} e' \ .$$

- If $e = e_0^*$, then we calculate by induction that

$$e = e_0^* \equiv_{\mathsf{R}} 1 + e_0 \cdot e_0^* \equiv_{\mathsf{R}} 1 + \sum_{e'_0 \in \iota(e'_0)} e'_0 \cdot e_0^* \equiv_{\mathsf{R}} \sum_{e' \in \iota(e)} e' \ . \qquad \square$$

Lemma 6.34. *Let $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$. The following hold.*

(i) For all $\mathsf{a} \in \Sigma$ and $e' \in \delta(e, \mathsf{a})$, we have $\mathsf{a} \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e$.

(ii) For all $\alpha \in \mathsf{At}$ and $e' \in \zeta(e, \alpha)$, we have $\alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e$.

Proof. We prove the second claim by induction on e; the first claim can be shown analogously. For the base, the claim holds vacuously if $e \in \{0,1\} \cup \Sigma$. When $e \in \mathsf{At}$, we have $e' = 1$ and $e = \alpha$. Hence, we conclude that $\alpha \cdot e' \equiv_{\mathsf{R}}^{\mathsf{contr}'} e$. For the inductive step, there are three cases to consider.

- If $e = e_0 + e_1$, then either $e' \in \zeta(e_0, \alpha)$ or $e' \in \zeta(e_1, \alpha)$; w.l.o.g., we assume the former. By induction, we then find that $\alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0 \leqq_{\mathsf{R}} e$.
- If $e = e_0 \cdot e_1$, then there are two cases to consider.
  - If $e' = e'_0 \cdot e_1$ for some $e'_0 \in \zeta(e_0, \alpha)$, then by induction we know that $\alpha \cdot e'_0 \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0$. It then follows that $\alpha \cdot e' = \alpha \cdot e'_0 \cdot e_1 \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0 \cdot e_1 = e$.
  - If $e' \in Z(e_0, e_1, \alpha) = \zeta(e_1, \alpha)$, then $\alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_1$, and one of two cases applies. If $e_0 \in \mathcal{F}_{\mathsf{R}}$, then $\alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_1 \equiv_{\mathsf{R}}^{\mathsf{contr}'} 1 \cdot e_1 \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0 \cdot e_1 = e$. Otherwise, $e'_0 \in \zeta(e_0, \alpha)$ with $e'_0 \in \mathcal{F}_{\mathsf{R}}$, then by induction $\alpha \cdot e'_0 \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0$. Hence, we can derive that

$$\alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} \alpha \cdot \alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} \alpha \cdot e'_0 \cdot \alpha \cdot e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0 \cdot e_1 \equiv_{\mathsf{R}}^{\mathsf{contr}'} e$$

- If $e = e_0^*$, then $e' = e'_0 \cdot e_0^*$ for some $e'_0 \in \zeta(e_0, \alpha)$. By induction, we know that $\alpha \cdot e'_0 \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0$. We can then derive that

$$\alpha \cdot e' = \alpha \cdot e'_0 \cdot e_0^* \leqq_{\mathsf{R}}^{\mathsf{contr}'} e_0 \cdot e_0^* \leqq_{\mathsf{R}} 1 + e_0 \cdot e_0^* \equiv_{\mathsf{R}} e_0^* \ . \qquad \square$$

To prove Lemma 6.35, we need three technical lemmas.

Lemma 6.A.1. *Let $e, f \in \mathcal{T}_\mathsf{R}(\Gamma)$, and let $s : \rho(e) \to \mathcal{T}$ be given by $s(e') = s_e(e') \cdot f$. Now s is the least $\rho(e)$-vector s such that for each $e' \in \rho(e)$, it holds that*

$$[e' \in \mathcal{F}_\mathsf{R}] \cdot f + \sum_{e'' \in \delta(e', a)} a \cdot s(e'') + \sum_{e'' \in \zeta(e', \alpha)} \alpha \cdot s(e'') \leqq_\mathsf{R} s(e') \ .$$

Proof. Suppose that s' is a $\rho(e)$-vector that satisfies the constraints above. Using Lemma 6.28, we can then calculate for any $e' \in \rho(e)$ as follows:

$$\begin{aligned} b_e(e') \cdot f + \sum_{e'' \in \rho(e)} M_e(e', e'') \cdot s'(e'') &\equiv_\mathsf{R} b_e(e') \cdot f + \sum_{e'' \in \rho(e)} \Big(\sum_{e'' \in \delta(e', \mathtt{a})} \mathtt{a} + \sum_{e'' \in \zeta(e', \alpha)} \alpha \Big) \cdot s'(e'') \\ &\equiv_\mathsf{R} b_e(e') \cdot f + \sum_{e'' \in \delta(e', \mathtt{a})} \mathtt{a} \cdot s'(e'') + \sum_{e' \in \zeta(e'', \alpha)} \alpha \cdot s'(e'') \\ &\leqq_\mathsf{R} s'(e') \end{aligned}$$

Since s is the least $\rho(e)$-vector for which this holds (c.f. Theorem 4.43), it follows that for $e' \in \rho(e)$ we have $s(e') \leqq s'(e')$. A similar derivation shows that s indeed satisfies the desired constraint. □

Lemma 6.A.2. *Let $e, f \in \mathcal{T}_\mathsf{R}(\Gamma)$. If $f' \in \rho(f)$ and $\rho(f) \subseteq \rho(e)$, then $s_f(f') \leqq_\mathsf{R} s_e(f')$.*

Proof. We fix a $\rho(f)$-vector s by choosing for $f' \in \rho(f)$ that $s(f') = s_e(f')$. By Lemma 6.A.1, we have for any $f' \in \rho(f)$ that the following holds:

$$[f' \in \mathcal{F}_\mathsf{R}] + \sum_{f'' \in \delta(f', \mathtt{a})} \mathtt{a} \cdot s(f'') + \sum_{f'' \in \zeta(f', \alpha)} \alpha \cdot s(f'') \leqq_\mathsf{R} s(f') \ .$$

Hence, s satisfies the system of linear equations obtained from f, and therefore s_f is a lower bound of s; the claim then follows. □

Lemma 6.A.3. *Let $e, f \in \mathcal{T}_\mathsf{R}(\Gamma)$. If $e' \in \rho(e)$ and $f' \in \iota(f)$, then $s_e(e') \cdot s_f(f') \leqq_\mathsf{R} s_{e \cdot f}(e' \cdot f)$.*

Proof. Using Lemmas 6.A.1, 6.A.2 and 6.31, we calculate for all $e' \in \rho(e)$:

$$\begin{aligned} &[e' \in \mathcal{F}_\mathsf{R}] \cdot s_f(f') + \sum_{e'' \in \delta(e', \mathtt{a})} \mathtt{a} \cdot s_{e \cdot f}(e'' \cdot f) + \sum_{e'' \in \zeta(e', \alpha)} \alpha \cdot s_{e \cdot f}(e'' \cdot f) \\ &\equiv_\mathsf{R} [e' \in \mathcal{F}_\mathsf{R}] \cdot \Big([f' \in \mathcal{F}_\mathsf{R}] + \sum_{f'' \in \delta(f', \mathtt{a})} \mathtt{a} \cdot s_f(f') + \sum_{f'' \in \zeta(f', \alpha)} \alpha \cdot s_f(f') \Big) \\ &\quad + \sum_{e'' \in \delta(e', \mathtt{a})} \mathtt{a} \cdot s_{e \cdot f}(e'' \cdot f) + \sum_{e'' \in \zeta(e', \alpha)} \alpha \cdot s_{e \cdot f}(e'' \cdot f) \\ &\leqq_\mathsf{R} [e' \cdot f \in \mathcal{F}_\mathsf{R}] + \sum_{g \in \delta(e' \cdot f, \mathtt{a})} \mathtt{a} \cdot s_{e \cdot f}(g) + \sum_{g \in \zeta(e' \cdot f, \alpha)} \alpha \cdot s_{e \cdot f}(g) \equiv_\mathsf{R} s_{e \cdot f}(e' \cdot f) \ . \end{aligned}$$

Hence, the $\rho(e)$-vector s where $s(e') = s_{e \cdot f}(e' \cdot f)$ satisfies the system obtained from e and $s_f(f')$. By Lemma 6.A.1, we find for all $e' \in \rho(e)$ that $s_e(e') \cdot s_f(f') \leqq_{\mathsf{R}} s_{e \cdot f}(e' \cdot f)$. □

Lemma 6.35. *If $e \in \mathcal{T}_{\mathsf{R}}(\Gamma)$ and $e' \in \rho(e)$, then $e' \leqq_{\mathsf{R}} s_e(e')$.*

Proof. We proceed by induction on e. In the base, there are four cases.

- If $e = 0$, then $\rho(e) = \emptyset$, and so the claim holds trivially.
- If $e = 1$, then $e' = 1$, and $e' = 1 = b_e(1) \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(1) = s_e(e')$.
- If $e = \mathfrak{a}$ for some $\mathfrak{a} \in \Gamma$, then either $e' = \mathfrak{a}$, or $e' = 1$. The case where $e' = 1$ can be argued as before. On the other hand, if $e' = \mathfrak{a}$, then we find that

$$e' = \mathfrak{a} \equiv_{\mathsf{R}}^{\mathsf{contr}'} \mathfrak{a} \cdot 1 \leqq_{\mathsf{R}}^{\mathsf{contr}'} \mathfrak{a} \cdot s_e(1) \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(\mathfrak{a}) = s_e(e')$$

For the inductive step, there are three cases

- If $e = e_0 + e_1$, then either $e' \in \rho(e_0)$ or $e' \in \rho(e_1)$; we assume the former without loss of generality. By induction, we have that $e' \leqq_{\mathsf{R}} s_{e_0}(e')$. By Lemma 6.A.2, we know that $s_{e_0}(e') \leqq_{\mathsf{R}} s_e(e')$, completing the proof.
- If $e = e_0 \cdot e_1$, then either $e' = e_0' \cdot e_1$ for some $e' \in \rho(e_0)$, or $e' \in \rho(e_1)$. In the former case, we find by induction as well as Lemmas 6.A.3 and 6.31 that

$$e' = e_0' \cdot e_1 \equiv_{\mathsf{R}} \sum_{e_1' \in \iota(e_1)} e_0' \cdot e_1' \leqq_{\mathsf{R}}^{\mathsf{contr}'} \sum_{e_1' \in \iota(e_1)} s_{e_0}(e_0') \cdot s_{e_1}(e_1') \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(e') \ .$$

  In the latter case, we find $e' \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(e')$ as in the case where $e = e_0 + e_1$.
- If $e = e_0^*$, then it suffices to treat the case where $e' = e_0' \cdot e$ with $e_0' \in \rho(e_0)$ only. By Lemma 6.A.3, we derive for any $e'' \in \iota(e)$ that $s_{e_0}(e_0') \cdot s_e(e') \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_{e_0 \cdot e}(e_0' \cdot e)$. By Lemma 6.A.2 and the observation that $\rho(e_0 \cdot e) \subseteq \rho(e)$, we furthermore have $s_{e_0 \cdot e}(e_0' \cdot e) \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(e_0' \cdot e)$. We then conclude by induction, Lemma 6.31 and the above that

$$e' = e_0' \cdot e \equiv_{\mathsf{R}} \sum_{e'' \in \iota(e)} e_0' \cdot e'' \leqq_{\mathsf{R}}^{\mathsf{contr}'} \sum_{e' \in \iota(e)} s_{e_0}(e_0') \cdot s_e(e') \leqq_{\mathsf{R}}^{\mathsf{contr}'} s_e(e_0' \cdot e) = s_e(e')$$ □

Further Work

We conclude the first half of this thesis by listing some questions that remain open, as well as possible avenues for work that builds upon the results presented in the previous chapters.

Complexity Our results were about decidability of semantic (and, by proxy, axiomatic) equivalence of sr-expressions. More granular analysis is necessary to see whether these problems can be solved efficiently, and pin down the appropriate complexity class. Some results about complexity of equivalence checking already exist; equivalence of rational expressions w.r.t. $\equiv$ and $\equiv^{\mathsf{kat}}$, for instance, is PSPACE-complete [SM73; CKS96]; other sets of hypotheses were analysed on the level of rational expressions in [DKP$^+$19]. Furthermore, equivalence of sr-expressions w.r.t. $\equiv^{\mathsf{exch}}$ is EXPSPACE-complete [BPS17]. In op. cit., deciding $\equiv$-equivalence is shown to be in EXPSPACE; the question of whether it is EXPSPACE-complete remains open. The reduction of exch to the empty set of hypotheses could be viewed as a reduction from the former problem to the latter problem. Unfortunately, this does show directly that equivalence of sr-expressions w.r.t. $\equiv$ is EXPSPACE-complete, because the sr-expressions $e{\downarrow}$ computed for each $e \in \mathcal{T}$ appears to grow quite rapidly, as a result of solving the sr-system, which suggests that the reduction is not polynomial.

Parallel Star The reduction of exch to the empty set of hypotheses worked for sr-expressions, and we noted that adding parallel star as an operator (moving to spr-expressions) prevents the existence of a *strong* reduction. However, we have no evidence to suggest that a (non-strong) reduction cannot exist, nor that $\equiv^{\mathsf{exch}}$ is incomplete w.r.t. equivalence of spr-expressions up to exch-closure. One possible way to tackle this problem could be to look at (ordered) bi-monoids [BÉ96; LW00]; finite bi-monoids relate to spr-expressions in the same way that finite monoids relate to rational expressions as described by the famous Myhill-Nerode theorem [Ner58]. Finite monoids can be used to come up with reductions for rational languages [KM14], and it would be interesting to see whether similar techniques apply to series-parallel rational languages.

Critical Sections The exchange law is a form of unbridled interleaving: if actions take place in sequence, then anything from a concurrent thread may happen in between. This led to problems when we combined hypotheses on assertions (from kat) with the exchange law. Programming languages have developed a mechanism to tame interleaving: *critical sections* protect a piece of code from being interleaved with actions inside other threads. Recent work has incorporated an operator to delineate critical sections into sr-expressions, along with several sets of hypotheses to reason about the resulting expressions [BP20]. It would be interesting to see whether this extension of sr-expressions works well with the hypotheses to reason about assertions from ckao.

Contextual equivalence A gsr-expression like $p \cdot \overline{p}$ describes pomsets of the form $\alpha \cdot \beta$, where $\alpha \leqq_{\mathsf{B}} p$ and $\beta \leqq_{\mathsf{B}} \overline{p}$, i.e., first the state of the machine is described by the atom α, and then by a *different* atom β, with nothing to mediate this change. The reason for this "spooky action at a distance" is that if $p \cdot \overline{p}$ had no semantics, then it would be equivalent to 0, which would mean that $p \cdot \overline{p} \parallel e$ would also be equivalent to 0 for any gsr-expression e, causing the problem in Fact 6.6. Thus, $p \cdot \overline{p}$ has no valid execution in isolation (because there is nothing to interleave with) but, in some contexts, it must give rise to pomsets that combine with others to form sensible executions. A way to get around this would be to extract the pomsets that make sense as an execution of a program in isolation; guarded pomsets [JM16] seem a reasonable candidate. This gives two ways of comparing a program: using the semantics $[\![-]\!]^{\mathsf{ckao}}$ encountered before, and using the "sensible" part of this semantics; the former would then correspond to equivalence of programs *in any context*, whereas the latter models equivalence *in isolation*. We conjecture that these are connected, in that equivalence w.r.t. the former is the largest congruence contained in equivalence w.r.t. the latter.

Partial Tests The law of excluded middle ($p \vee \overline{p} \equiv_{\mathsf{B}} \top$), which says that an assertion either does or does not hold, may not be valid in a setting where threads have only partial knowledge of the state of the machine. In this case, it could be that the veracity of p cannot be ascertained, as it depends on a part of the state not accessible by the current thread, and therefore there may not be any positive evidence to support it. Jipsen and Moshier [JM16] studied assertions related by a weakened version of the axioms of Boolean algebra which does not include the law of excluded middle, known as the axioms of *pseudo-complemented distributive lattices* (*PCDLs*) [BD74]. We have recently explored an extension of ckao to this weaker setting, including assertions about variables of the form $f = n$ and primitive actions that modify those variables, i.e., of the form $f \leftarrow n$ [WBD+20]. In particular, we think this framework, which relies heavily on the use of hypotheses, could be used to prove properties of concurrency in programming languages through litmus tests [AMS+11].

Star-continuity If the axioms of rational expressions that build $\equiv_{\mathsf{R}}$ are augmented with the (infinitary) star-continuity axiom, one can obtain a generic completeness result [DKP^{+}19], i.e., for rational expressions e and f and a set of hypotheses H it would hold that $e \equiv_{\mathsf{R}}^{H} f$ if and only if $\llbracket e \rrbracket^{\langle H \rangle} = \llbracket f \rrbracket^{\langle H \rangle}$. We would like to know whether the same applies to sr-expressions, or indeed spr-expressions with a similar continuity axiom for the parallel star. If this turns out to be true, it may open up the possibility of axiomatising synchronous rational languages [Pri10] without the Salomaa-style unique fixpoint axioms used in [WBK^{+}19], but with the star-continuity axiom.

Quasi-hypotheses The unique fixpoint axioms proposed by Salomaa cannot be captured by the framework of hypotheses, because they are quasi-equations rather than equations, i.e., they require some other precondition to hold. We wonder whether hypotheses can be generalised to include such quasi-hypotheses, and whether useful reductions can still be obtained. In particular, we wonder whether a similar generic completeness theorem can be derived, i.e., whether augmenting the axioms for series-rational expressions with star continuity and quasi-hypotheses gives a sound and complete proof principle for equivalence under a supposed closure w.r.t. quasi-hypotheses.

Guarded control The traditional control flow structures of **if** p **then** e **else** f and **while** p **do** e can be encoded using assertions, as $p \cdot e + \overline{p} \cdot f$ and $(p \cdot e)^{*} \cdot \overline{p}$ respectively. Alternatively, we can forego the use of non-deterministic composition and the Kleene star as operators, and include guarded composition as a first-class citizen [KT08]. The benefit of this shift to *guarded Kleene algebra with tests* (*GKAT*) is that it significantly reduces the complexity of deciding equivalence [SFH^{+}20]. We would extend GKAT with hypotheses, in hopes of finding similarly efficient algorithms to mechanise equivalence of programs. Orthogonally, perhaps GKAT can be extended with concurrent composition in the same way that we weakened KAT, and then extended with concurrent composition.

Bisimilarity One could argue that equivalence in (concurrent) Kleene algebra is too coarse; for instance, we do not distinguish between expressions like $(e + f) \cdot g$ and $e \cdot g + f \cdot g$, while the latter makes a choice between e and f before executing g, whereas the former does not. Such a distinction is made in process algebra [Hoa78; Mil80], where programs are not compared based on the runs that they give rise to, but rather based on whether one program can replicate the behaviour of another and vice versa, using *bisimulation*. We want to investigate whether hypotheses can be applied to equational axiomatisations of bisimulation (see, e.g., [AF06; AI07; BLB19]) as well, and whether the well-known results about Kleene algebra can be recovered from such a framework. The study of *probabilistic Kleene algebra* [MW05] already includes some weakening of the distributive law.

Network programming NetKAT [AFG$^+$14] is a programming language that can be used to specify and reason about packet routing in a software-defined network, based on Kleene algebra with tests. It features a sound and complete axiomatisation [AFG$^+$14] as well as an efficient decision procedure based on coalgebra [FKM$^+$15]. We would like to apply our insights on tests and concurrency to NetKAT, in order to derive an extension that allows reasoning about concurrent packet processing, which could either take place within a device, or distributed among different devices. For instance, it would be interesting to specify and reason about the behaviour of a network where inbound packets are filtered concurrently by a firewall and an intrusion detection system.

Logic Kleene algebra with tests is closely connected to propositional Hoare logic [Koz00]. More specifically, one can show that every propositionally valid Hoare triple corresponds to a valid equation in KAT. We would like to investigate whether the propositional versions of concurrent extensions of this logic, such as concurrent separation logic [Bro07; OHe07], can be similarly connected to concurrent Kleene algebra with observations.

Part II

Automata

Chapter 7

Pomset Automata

Kleene's theorem states that rational expressions correspond exactly to languages accepted by finite automata [Kle56]. Such a correspondence is extremely useful, because it allows us to shift perspectives from the denotational description to the operational description of a rational language (or vice versa) when reasoning. For instance, Kleene's theorem was crucial in axiomatising equivalence of rational expressions [Sal66; Con71; Koz94]. Furthermore, an operational description of semantics also lends itself to mechanisation: it is relatively easy to decide whether a finite automaton accepts a given word, or whether two finite automata accept the same language.

Given these benefits, it seems worthwhile to develop a similar correspondence for series-rational expressions. In fact, several such correspondences exist in the literature; they can be classified by the type of operational formalism to describe pomset languages. First, we have approaches based on *Petri nets* [Pet62]. Grabowski provided a two-way correspondence between 1-*safe* Petri nets and pomset languages denoted by a generalisation of sr-expressions [Gra81]. Lodaya et al. later proved that series-rational languages correspond precisely to the series-parallel parts of these 1-safe Petri nets [LRR03]. Brunet et al. showed that an extension of Grabowski's construction to translate an sr-expression into a Petri net can be used to decide equivalence of sr-expressions [BPS17].

Other constructions are based on automata theory. Ésik and Németh proposed *higher dimensional automata* and related these to sr-expressions, for a language model were parallel composition is non-commutative [ÉN04]. Lodaya and Weil invented *branching automata*, which are finite automata augmented with two types of transition that specify where computation can *fork* and *join* [LW00]. Later, Jipsen and Moshier proposed their own type of branching automaton, which specifies where computations can join after having forked from a predesignated state [JM16].

Petri automata are a formalism situated between these approaches, proposed by Brunet and Pous [BP17]. On the one hand, Petri automata can be thought of as 1-safe Petri nets with a particular structure; on the other hand, they correspond closely to Lodaya and Weil's branching automata. The main difference between Petri automata and the formalisms discussed above is that they are aimed at recognising *series-parallel graphs*. Superficially, series-parallel graphs can be thought of as sp-pomsets where the edges rather than the vertices are labelled; however, series-parallel graphs may also contain cycles, and are therefore a strictly richer model.

In this chapter, we will define *pomset automata*, another operational model for pomset languages. We will identify a class of pomset automata called *fork-acyclic pomset automata*, and show that languages accepted by these correspond precisely to series-rational languages, thus establishing a Kleene-like theorem. More accurately, we will show that a series-rational expression can be translated into a fork-acyclic pomset automaton accepting the same language, and that given a fork-acyclic pomset automaton we can obtain a series-rational expression describing its language.

Our work differs from previous investigations on three points.

- First, previous translations from series-rational expressions to automata or Petri nets used an inductive approach similar to that of Thompson [Tho68]. In contrast, our translation is in the style of Brzozowski [Brz64] and Antimirov [Ant96]. The advantage of this approach is that it allows one to construct the automaton lazily, i.e., on the fly. This can be advantageous for algorithms that are structured in such a way as to explore the state space starting from some initial state, and may terminate early, preventing exploration of the whole state space.

- Second, the operational formalisms are typically more expressive than series-rational expressions and, as a result, the translation back to series-rational expressions requires an additional condition on the automaton or Petri net to be correct. The problem is that, in existing work, this translation is typically phrased in semantic terms, which makes it unclear whether membership of this class of automata is decidable. On the other hand, the fork-acyclicity property that we require of pomset automata is presented structurally, and is in fact decidable.

- Lastly, in contrast with other constructions on automata, we derive (in Chapter 8) an algorithm that decides whether two states in a pomset automaton accept the same pomset language. By proxy, this gives an alternative proof that equivalence of series-rational expressions is decidable — supplementing earlier proofs by other authors [LS14; BPS17]. More generally, our approach can be seen as a generalisation of Brzozowski and Tamm's algorithm, which calculates the "atoms" of a finite automaton by reverse determinisation [BT14].

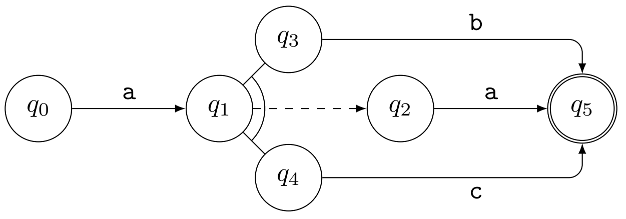


Figure 7.1: Pomset automaton accepting $\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c}) \cdot \mathtt{a}$.

7.1 Automata model

Imagine writing a program with fork/join concurrency. When the program reaches a point where parallel computation is desirable, one could imagine saying "run these functions in parallel, and when both are done, resume computation here". Pomset automata reflect this intuition: they have transitions that allow computation to be forked into several states; once each of those threads has reached an accepting state, computation resumes elsewhere. More formally, we have the following.

Definition 7.1 (Pomset automaton)**.** A *pomset automaton* (PA) is a tuple $\langle Q, F, \delta, \gamma \rangle$ where

- Q is a set of *states*, with $F \subseteq Q$ the *accepting states*, and
- $\delta : Q \times \Sigma \to 2^Q$ the *sequential transition function*, and
- $\gamma : Q \times \mathbb{M}(Q) \to 2^Q$ the *parallel transition function.*[1]

Lastly, for all $q \in Q$, there are only finitely many $\phi \in \mathbb{M}(Q)$ such that $\gamma(q, \phi) \neq \emptyset$.

In the above, $q' \in \delta(q, \mathtt{a})$ means that A may transition from q to q' if it performs the action $\mathtt{a}$. The parallel transition function implements forking and joining: $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$ should be interpreted to mean that, in state q, the automaton may fork into the multiset of states $\{\!|r_1, \dots, r_n|\!\}$, and, as soon as each of these reaches a (possibly distinct) accepting state, may resume in q'.

Remark 7.2. One can think of PAs as the dual to Jipsen and Moshier's branching automata [JM16]. Branching automata say where computation can join after branching off from a given state, while PAs specify where computation can fork, provided that it commits to resume at a certain place.

We can draw finite PAs as in Figure 7.1. A state is represented by a vertex, doubly circled when the state is accepting. Edges represent transitions: the edge labelled $\mathtt{a}$ between q_0 and q_1 encodes that $q_1 \in \delta(q_0, \mathtt{a})$, and the multi-ended edge signifies that $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$. A PA is called *finite* if it has finitely many states. A finite PA has finitely many transitions, and can therefore be drawn.

[1]Recall that $\mathbb{M}(Q)$ is the set of finite multisets with elements in Q.

For the remainder of this section, we fix a PA $A = \langle Q, F, \delta, \gamma \rangle$. We proceed to define how one can "read" an sp-pomset U while transitioning from a state q to a state q', matching the intuition of the parallel transition function. This information is encoded in a ternary relation between states, pomsets, and states, called the *run relation.* If a state q is related to a pomset U and another state q', it means that, starting in state q, we can read the pomset U to end up in state q'. The pomsets that we can read to reach an accepting state form the language accepted by a state.

Definition 7.3 (Runs and language)**.** We define the *run relation* $\rightarrow_A \subseteq Q \times \mathsf{SP} \times Q$ as the smallest set satisfying the following rules, where we abbreviate $\langle q, U, q \rangle \in \rightarrow_A$ by writing $q \xrightarrow{U}_A q'$.

$$\frac{}{q \xrightarrow{1}_A q} \qquad \frac{q' \in \delta(q, \mathtt{a})}{q \xrightarrow{\mathtt{a}}_A q'} \qquad \frac{q \xrightarrow{U}_A q'' \qquad q'' \xrightarrow{V}_A q'}{q \xrightarrow{U \cdot V}_A q'} \qquad \frac{\forall 1 \leq i \leq n.\ q_i \xrightarrow{U_i}_A q_i' \in F \qquad q' \in \gamma(q, \{\!|q_1, \dots, q_n|\!\})}{q \xrightarrow{U_1 \parallel \cdots \parallel U_n}_A q'}$$

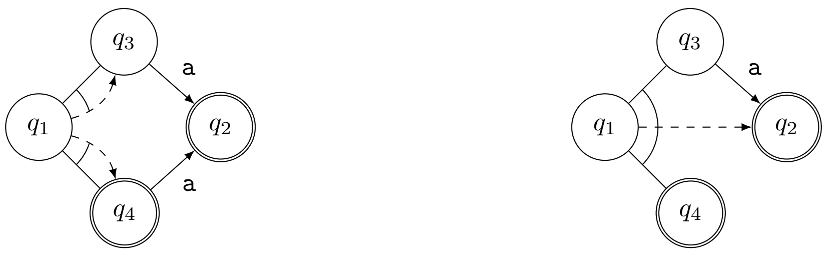
The *language* accepted by $q \in Q$, denoted $L_A(q)$, is the set $\left\{U \in \mathsf{SP} \ : \ q \xrightarrow{U}_A q' \in F\right\}$.

Given an element of the run relation, a proof tree witnessing that this triple occurs in $\rightarrow_A$ contains structural information about how the pomset automaton read the pomset, i.e., the order the states were visited, which fork transitions were used, et cetera. Such a proof tree can therefore be called a *run* of the pomset automaton. We shall often abuse this nomenclature and refer to individual elements of the run relation as runs, with their underlying proof tree implicitly present.

Example 7.4. Suppose A is the PA in Figure 7.1. Then, we have that $q_0 \xrightarrow{\mathtt{a}}_A q_1$, $q_2 \xrightarrow{\mathtt{a}}_A q_5$, $q_3 \xrightarrow{\mathtt{b}}_A q_5$, and $q_4 \xrightarrow{\mathtt{c}}_A q_5$. From the latter two runs and the fact that $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$, it follows that $q_1 \xrightarrow{\mathtt{b} \parallel \mathtt{c}}_A q_2$ by the last rule. By applying the third rule to this run and the first two runs above, we find $q_0 \xrightarrow{\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c}) \cdot \mathtt{a}}_A q_5$. Since $q_5 \in F$, we have that $\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c}) \cdot \mathtt{a} \in L_A(q_0)$.

Remark 7.5. Due to the non-deterministic nature of the transition functions, there may be more than one way to read a pomset starting in a state. In principle, it is possible to determinise these transition functions using a powerset construction [RS59]. The problem, however, is that even if every transition has at most one target, there is still an implicit type of non-determinism. For instance, if A is the PA in Figure 7.2a, then $q_1 \xrightarrow{\mathtt{a}}_A q_2$ and $q_1 \xrightarrow{\mathtt{a}}_A q_3$. Since we are going to have to deal with such ambiguity either way, we leave the non-determinism in the transition functions in place. In fact, we will exploit the flexibility of non-determinism in constructions to come.

It is useful to distinguish runs based on the rules that induce them. To this end, we establish the following terminology for $q, q' \in Q$ and $U \in \mathsf{SP}$.



(a) A PA with implicit non-determinism.

(b) A PA with run confusion.

Figure 7.2: Pomset automata that exhibit unexpected effects of parallel transitions.

- If $q \xrightarrow{U}_A q'$ follows by an application of the first rule, we speak of a *trivial run.*
- If $q \xrightarrow{U}_A q'$ has a derivation where the second rule is applied last, it is a *sequential unit run.*
- If $q \xrightarrow{U}_A q'$ is a consequence of the last rule, this run is a *parallel unit run.*
- The sequential and parallel unit runs are collectively referred to as *unit runs.*
- If $q \xrightarrow{U}_A q'$ is a result of the third rule, i.e., there exist non-empty $U_1, U_2 \in \mathsf{SP}$ and $q'' \in Q$ such that $q \xrightarrow{U_1}_A q''$ as well as $q'' \xrightarrow{U_2}_A q'$, then $q \xrightarrow{U}_A q'$ is known as a *composite run.*

By definition of $\rightarrow_A$, each run falls into at least one (and possibly more) of these categories.

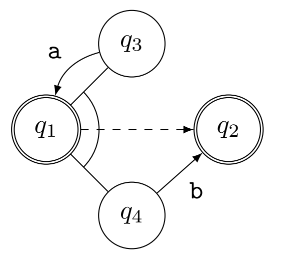
Example 7.6. Returning to Example 7.4 above, $q_0 \xrightarrow{\mathtt{a}}_A q_1$ is a sequential unit run, and $q_1 \xrightarrow{\mathtt{b} \parallel \mathtt{c}}_A q_2$ is a parallel unit run; $q_5 \xrightarrow{1}_A q_5$ is a trivial run. Lastly, $q_0 \xrightarrow{\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c}) \cdot \mathtt{a}}_A q_5$ is a composite run.

Remark 7.7. The kind of a run is not uniquely determined by the kind of pomset that labels it, that is, not all runs labelled by parallel pomsets are parallel unit runs. For instance, if A is the PA in Figure 7.2b, then we can construct the parallel unit run $q_1 \xrightarrow{\mathtt{a}}_A q_2$, even though $\mathtt{a}$ is not parallel. Similarly, not every run labelled by the empty pomset is trivial. For instance, if $q, q' \in Q$ such that $q' \in \gamma(q, \boxslash)$, then $q \xrightarrow{1}_A q'$.[2] We reckon with this kind of confusion between run types in Chapter 8.

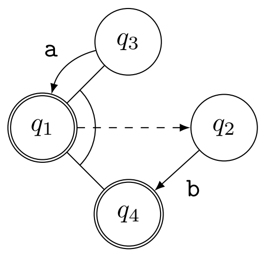
When reasoning about runs in a pomset automaton, it is often useful to split up a composite run, and then split up any composite runs that result from that, until only unit runs are left. Any trivial runs that are found in this process can be omitted, because they do not change the pomset accepted. We can iterate this process, as witnessed by the following lemma.

Lemma 7.8. *Let $q \xrightarrow{U}_A q'$. There exist $q = q_0, \dots, q_\ell = q' \in Q$ and $U_1, \dots, U_\ell \in \mathsf{SP}$, such that $U = U_1 \cdots U_\ell$, and for all $1 \leq i \leq \ell$ we have that $q_{i-1} \xrightarrow{U_i}_A q_i$ is a unit run.*

[2]Recall that $\boxslash$ is the empty multiset.



(a) A PA with unbounded parallelism.



(b) A PA accepting $\mathtt{a}^n \cdot \mathtt{b}^n$.

Figure 7.3: Some problematic pomset automata.

The minimal ℓ for a given run as obtained above is known as the *length* of the run. We shall use the length of a run as an inductive handle on several occasions.

7.1.1 Fork-acyclicity

Pomset automata are a powerful model of computation. In particular, they can be used to accept pomset languages that are not series-rational, either because they have unbounded width, or because they are in fact languages of words that cannot be described by a rational expression. This is a problem if we want to describe the language accepted by a state in PA using an sr-expression.

Example 7.9. Suppose A is the PA in Figure 7.3a. We then find that $q_4 \xrightarrow{\mathtt{b}}_A q_2$ and $q_3 \xrightarrow{\mathtt{a}}_A q_1$. Since $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$, we know that $q_1 \xrightarrow{\mathtt{a} \parallel \mathtt{b}}_A q_2$. However, because $q_3 \xrightarrow{\mathtt{a}}_A q_1$, it follows that $q_3 \xrightarrow{\mathtt{a} \cdot (\mathtt{a} \parallel \mathtt{b})}_A q_2$, and hence we also find that $q_1 \xrightarrow{\mathtt{a} \cdot (\mathtt{a} \parallel \mathtt{b}) \parallel \mathtt{b}}_A q_2$. This pattern can be repeated indefinitely, leading to an unbounded number of forks in the construction of runs originating from q_1. In addition, we note that $L_A(q_1)$ is of unbounded depth, and therefore not series-rational [LS14].

Example 7.10. Suppose A is the PA in Figure 7.3b. In this PA, the state q_1 accepts the non-rational language $L = \{\mathtt{a}^n \cdot \mathtt{b}^n \;:\; n \in \mathbb{N}\}$ [RS59; BPS61]. To show $L \subseteq L_A(q_1)$, we claim that if $n \in \mathbb{N}$, then $q_1 \xrightarrow{\mathtt{a}^n \cdot \mathtt{b}^n} q$ with $q \in F$. To see this, first note that for $n = 0$, the claim holds trivially if we choose $q = q_1$. If the claim holds for n, then note that since $q_3 \xrightarrow{\mathtt{a}} q_1$, we have $q_3 \xrightarrow{\mathtt{a}^{n+1} \cdot \mathtt{b}^n} q$ with $q \in F$. Since $q_4 \xrightarrow{1} q_4 \in F$ and $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$, also $q_1 \xrightarrow{\mathtt{a}^{n+1} \cdot \mathtt{b}^n} q_2$; since $q_2 \xrightarrow{\mathtt{b}} q_4$, we conclude that $q_1 \xrightarrow{\mathtt{a}^{n+1} \cdot \mathtt{b}^{n+1}} q_4 \in F$. The proof of the converse inclusion is left as an exercise to the reader.

To prevent this excessive amount of expressive power, we need to put a structural restriction on PAs. Indeed, earlier automata models for sr-expressions had to apply similar restrictions [LW00; JM16]. The common factor in these cases is the behaviour that can occur when a state q can fork into a state q', which can start a run that somehow involves the original state q, through a series of transitions and forks. Hence, the first step in inhibiting this kind of mutual dependency across forks is to get a handle on the states that can be involved in constructing a run that originates in a given state, by transitioning to that state, by forking into it, or some combination of the two.

Definition 7.11 (Support relation)**.** We define $\preceq_A$ as the smallest preorder on Q s.t. for $q \in Q$:

$$\frac{\mathtt{a} \in \Sigma \qquad q' \in \delta(q, \mathtt{a})}{q' \preceq_A q} \qquad \frac{\phi \in \mathbb{M}(Q) \qquad q' \in \gamma(q, \phi)}{q' \preceq_A q} \qquad \frac{r \in \phi \in \mathbb{M}(Q) \qquad \gamma(q, \phi) \neq \emptyset}{r \preceq_A q}$$

We refer to $\preceq_A$ as the *support relation* of A. This relation in turn gives rise to the *strict support relation* $\prec_A$, which is the strict order where $q' \prec_A q$ holds if $q' \preceq_A q$ and $q \not\preceq_A q'$.

Example 7.12. Returning to the PA A in Figure 7.3a, we see that $q_5 \preceq_A q_4$, since $q_5 \in \delta(q_4, \mathtt{b})$ (by the first rule). Since $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$, it follows that $q_1 \preceq_A q_1$ (by the second rule) as well as $q_3, q_4 \preceq_A q_1$ (by the third rule). By transitivity, it then follows that $q_5 \preceq_A q_1$.

Intuitively, if q' is necessary to establish some run originating from q, then $q' \preceq_A q$; hence, we say that q' *supports* q. In particular, if $r \preceq_A q$ because there exists a $\phi \in \mathbb{M}(Q)$ with $r \in \phi$ and $\gamma(q, \phi) \neq \emptyset$, then r serves as the start of one or more threads that q may fork into, and we say that r is a *fork target* of q. Support can be mutual; such a two-way dependency need not be a problem: for instance when $q' \in \delta(q, \mathtt{a})$ and $q \in \delta(q', \mathtt{b})$; consequently, $\preceq_A$ need not be antisymmetric.

To break fork cycles, we can define a restriction that avoids infinitely nested forks, by stipulating a state cannot support any of its fork targets. Formally, this restriction is as follows.

Definition 7.13 (Fork-acyclicity)**.** We say that A is *fork-cyclic* if for some $q, r \in Q$ such that r is a fork target of q, we have that $q \preceq_A r$; A is *fork-acyclic* if it is not fork-cyclic.

If A is finite and fork-acyclic, we write $D_A(q)$ for the *depth of* $q \in Q$ *in* A, which is the maximum n such that there exist $q_1, \dots, q_n \in Q$ with $q_1 \prec_A q_1 \prec_A \cdots \prec_A q_n = q$. We furthermore write D_A for the *depth of* A, which is defined as the maximum of $D_A(q)$ for all states q.

Example 7.14. Returning to the PA in Figure 7.3a, we see that q_3 is a fork target of q_1, while $q_1 \preceq_A q_3$. Hence, this PA is fork-cyclic. On the other hand, the PA in Figure 7.2b is fork-acyclic, because neither q_3 nor q_4 is supported by q_1; it has depth 2, since $q_3 \prec_A q_1$.

7.1.2 Implementation

In some cases, it is useful to restrict a pomset automaton to a subset of its states. For such a restriction to be sound, we need to be sure that the transition functions in this subset require only other states in the subset. To that end, the following notion is helpful.

Definition 7.15 (Support)**.** We say that $Q' \subseteq Q$ is *support-closed* if for all $q \in Q'$ with $q' \preceq_A q$ we have $q' \in Q'$. The *support* of $q \in Q$, denoted $\pi_A(q)$, is the smallest support-closed set containing q.

Example 7.16. In Figure 7.3a, the set $\{q_2, q_4\}$ is support-closed and is in fact the support of q_2. The set $\{q_3\}$ is not, since $q_1 \preceq_A q_3$. The support of q_1 is given by the set of all states.

Our main interest is finite pomset automata. It will also be convenient to relax this property and speak of pomset automata with infinitely many states. Such a PA may yet allow an unbounded number of nested forks and accept non-series-rational languages. We thus want to ensure that we can restrict such a PA to a finite fragment of interest. The following helps with that.

Definition 7.17 (Bounded)**.** When $\pi_A(q)$ is finite for all $q \in Q$, we say that A is *bounded*.

In the sequel, we will perform a number of transformations on automata to enforce desirable properties. To ensure correctness, we will require these constructions to transform an automaton A into an automaton A' that can accept the same languages as A, while preserving properties of A.

Definition 7.18 (Implementation)**.** Let $A = \langle Q, F, \delta, \gamma \rangle$ and $A' = \langle Q', F', \delta', \gamma' \rangle$ be pomset automata. We say that A' *implements* A if the following hold:

(i) $Q \subseteq Q'$ such that if $q \in Q$, then $L_A(q) = L_{A'}(q)$, and

(ii) if A is fork-acyclic, then so is A'.

We can restrict a PA to a support-closed subset, as follows

Definition 7.19. Let $Q' \subseteq Q$ be support-closed. We write $A[Q']$ for the PA $\langle Q', F \cap Q', \delta', \gamma' \rangle$, where $\delta' : Q' \times \Sigma \to 2^{Q'}$ and $\gamma' : Q' \times \mathbb{M}(Q') \to 2^{Q'}$ are the appropriate restrictions to Q', i.e.,

$$\delta'(q, \mathtt{a}) = \delta(q, \mathtt{a}) \qquad\qquad \gamma'(q, \phi) = \gamma(q, \phi)$$

Note that, because Q' is support-closed, the restrictions δ' and γ' are well-defined. For instance, if $q \in Q'$ and $\mathtt{a} \in \Sigma$, then $\delta'(q, \mathtt{a}) = \delta(q, \mathtt{a}) \subseteq Q'$, since if $q' \in \delta(q, \mathtt{a})$, then $q' \preceq_A q$, and thus $q' \in Q'$.

The restriction of a PA to a support-closed set of states preserves its languages, as well as properties such as fork-acyclicity and finiteness. More precisely, we have the following.

Lemma 7.20. *If Q' is support-closed, then $A[Q']$ implements A. If A is bounded, then so is $A[Q']$.*

Given a state q in a bounded PA A, we can restrict A to $A[\pi_A(q)]$ to obtain a PA that accepts $L_A(q)$. Since A is bounded, we have that $\pi_A(q)$ is finite, and therefore $A[\pi_A(q)]$ is also finite.

7.2 Expressions to automata

We proceed to show how, given an sr-expression e, we can obtain a fork-acyclic and finite PA, where some state accepts $[\![e]\!]$. By Lemma 7.20, it suffices to find a *bounded* and fork-acyclic PA where every sr-expression e is a state that accepts $[\![e]\!]$. We will craft the transition functions of this pomset automaton such that, if we start in $e \in \mathcal{T}$ and read the pomset U to arrive in e', then $[\![e']\!]$ contains pomsets V such that $U \cdot V \in [\![e]\!]$. This methodology is a generalisation of Antimirov's (partial) derivatives of rational expressions [Ant96], which is itself a variation on Brzozowski's derivatives [Brz64]; for this reason, we refer to the transition functions on expressions as *derivatives*.

Convention 7.21. Let $e \in \mathcal{T}$ and $S \subseteq \mathcal{T}$. We use $e \star S$ to denote S when $e \in \mathcal{F}$, and $\emptyset$ otherwise.

Definition 7.22 (Derivatives)**.** We define $\delta_{\mathsf{SR}} : \mathcal{T} \times \Sigma \to 2^{\mathcal{T}}$ and $\gamma_{\mathsf{SR}} : \mathcal{T} \times \mathbb{M}(\mathcal{T}) \to 2^{\mathcal{T}}$ inductively.

$$\begin{aligned}
\delta_{\mathsf{SR}}(0, \mathtt{a}) &= \emptyset & \gamma_{\mathsf{SR}}(0, \phi) &= \emptyset \\
\delta_{\mathsf{SR}}(1, \mathtt{a}) &= \emptyset & \gamma_{\mathsf{SR}}(1, \phi) &= \emptyset \\
\delta_{\mathsf{SR}}(\mathtt{b}, \mathtt{a}) &= \{1 \ : \ \mathtt{a} = \mathtt{b}\} & \gamma_{\mathsf{SR}}(\mathtt{b}, \phi) &= \emptyset \\
\delta_{\mathsf{SR}}(e + f, \mathtt{a}) &= \delta_{\mathsf{SR}}(e, \mathtt{a}) \cup \delta_{\mathsf{SR}}(f, \mathtt{a}) & \gamma_{\mathsf{SR}}(e + f, \phi) &= \gamma_{\mathsf{SR}}(e, \phi) \cup \gamma_{\mathsf{SR}}(f, \phi) \\
\delta_{\mathsf{SR}}(e \cdot f, \mathtt{a}) &= \delta_{\mathsf{SR}}(e, \mathtt{a}) \fatsemi f \ \cup \ e \star \delta_{\mathsf{SR}}(f, \mathtt{a}) & \gamma_{\mathsf{SR}}(e \cdot f) &= \gamma_{\mathsf{SR}}(e, \phi) \fatsemi f \ \cup \ e \star \gamma_{\mathsf{SR}}(f, \phi) \\
\delta_{\mathsf{SR}}(e \parallel f, \mathtt{a}) &= \emptyset & \gamma_{\mathsf{SR}}(e \parallel f, \phi) &= \{1 \ : \ \phi = \{\!|e, f|\!\}\} \\
\delta_{\mathsf{SR}}(e^*, \mathtt{a}) &= \delta_{\mathsf{SR}}(e, \mathtt{a}) \fatsemi e^* & \gamma_{\mathsf{SR}}(e^*, \phi) &= \gamma_{\mathsf{SR}}(e, \phi) \fatsemi e^*
\end{aligned}$$

We can now define our pomset automaton that operates on sr-expressions. Since the sr-expressions that accept the empty pomset are in $\mathcal{F}$, we choose those as the accepting states.

Definition 7.23. The *(series-rational) syntactic PA*, denoted by A_{SR}, is the PA $\langle \mathcal{T}, \mathcal{F}, \delta_{\mathsf{SR}}, \gamma_{\mathsf{SR}} \rangle$.

We simplify subscripts, writing $\to_{\mathsf{SR}}$ instead of $\to_{A_{\mathsf{SR}}}$, and so forth.

Example 7.24. We have drawn part of the syntactic PA, specifically the support of $\mathtt{a} \cdot \mathtt{b}^* \parallel \mathtt{c}$, in Figure 7.4. There, we see that $1 \cdot \mathtt{b}^*$ is an accepting state, because $1, \mathtt{b}^* \in \mathcal{F}$. The sequential transitions are generated by δ_{SR}; for instance, $1 \cdot \mathtt{b}^* \in \delta_{\mathsf{SR}}(\mathtt{a} \cdot \mathtt{b}^*, \mathtt{a})$, because $1 \in \delta_{\mathsf{SR}}(\mathtt{a}, \mathtt{a})$ and $\delta_{\mathsf{SR}}(\mathtt{a}, \mathtt{a}) \fatsemi \mathtt{b}^* \subseteq \delta_{\mathsf{SR}}(\mathtt{a} \cdot \mathtt{b}^*, \mathtt{a})$; also, $1 \cdot \mathtt{b}^* \in \delta_{\mathsf{SR}}(1 \cdot \mathtt{b}^*, \mathtt{b})$, because $1 \in \delta_{\mathsf{SR}}(\mathtt{b}, \mathtt{b})$, and $\delta_{\mathsf{SR}}(\mathtt{b}, \mathtt{b}) \fatsemi \mathtt{b}^* \subseteq \delta_{\mathsf{SR}}(1 \cdot \mathtt{b}^*, \mathtt{b})$. Lastly, $1 \in \gamma_{\mathsf{SR}}(\mathtt{a} \cdot \mathtt{b}^* \parallel \mathtt{c}, \{\!|\mathtt{a} \cdot \mathtt{b}^*, \mathtt{c}|\!\})$ by definition of γ_{SR}.

The remainder of this section is dedicated to show that the syntactic PA fulfills our objectives, i.e., that it is bounded and fork-acyclic, and that each state e accepts $[\![e]\!]$.

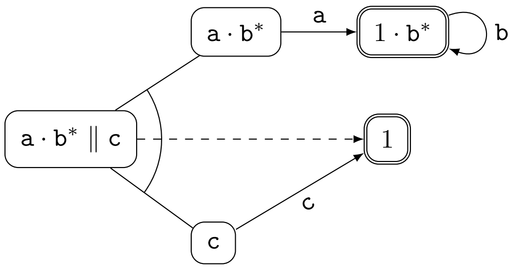


Figure 7.4: Part of the series-rational syntactic pomset automaton.

7.2.1 The languages accepted by the syntactic PA

To show that the syntactic PA accepts the right languages, we begin by analysing the runs that its transition structure allows. Specifically, given a run of A_{SR} that starts in $e \in \mathcal{T}$ and ends in a $f \in \mathcal{F}$, we show that we can obtain one or more runs starting in subexpressions of e. Conversely, we show how to construct runs to an accepting state if we have runs to accepting states for each of the operands that make up e. First, we show how this works for sums.

Lemma 7.25. *Let $e_1, e_2 \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists an $f \in \mathcal{F}$ such that $e_1 + e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) There exists an $f \in \mathcal{F}$ such that $e_1 \xrightarrow{U}_{\mathsf{SR}} f$ or $e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

Next, we show how a run that starts in a sequential composition and reaches an accepting state gives rise to runs that originate in the operands, and reach an accepting state too — and vice versa.

Lemma 7.26. *Let $e_1, e_2 \in \mathcal{T}$, $U \in \mathsf{SP}$, and $\ell \in \mathbb{N}$. The following are equivalent:*

(i) There exists an $f \in \mathcal{F}$ such that $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f$ of length ℓ.

(ii) $U = U_1 \cdot U_2$, and there exist $f_1, f_2 \in \mathcal{F}$ with $e_i \xrightarrow{U_i}_{\mathsf{SR}} f_i$ for $i \in \{1, 2\}$, of total length at most ℓ.

We can also relate runs originating in a parallel composition to runs from their operands.

Lemma 7.27. *Let $e_1, e_2 \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e_1 \parallel e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) $U = U_1 \parallel U_2$ and there exist $f_1, f_2 \in \mathcal{F}$ such that $e_i \xrightarrow{U_i}_{\mathsf{SR}} f_i$ for $i \in \{1, 2\}$.

For the last such lemma, we consider sr-expressions with the Kleene star as the topmost operator. Here, we show that a run originating in such an sr-expression to an accepting state gives rise to a number of runs, each of which originates in the starred expression, reaching an accepting state.

Lemma 7.28. *Let $e \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e^* \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) $U = U_1 \cdots U_n$, such that for $1 \leq i \leq n$ there exists $f_i \in \mathcal{F}$ with $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$.

These deconstruction and reconstruction lemmas combine to prove the equations claimed by the lemma below; deconstruction of a run proves the inclusion from left to right, whereas construction of a run can be used to show the inclusion from right to left.

Lemma 7.29. *Let $e, f \in \mathcal{T}$. The following hold:*

$$L_{\mathsf{SR}}(e + f) = L_{\mathsf{SR}}(e) + L_{\mathsf{SR}}(f) \qquad L_{\mathsf{SR}}(e^*) = L_{\mathsf{SR}}(e)^*$$
$$L_{\mathsf{SR}}(e \cdot f) = L_{\mathsf{SR}}(e) \cdot L_{\mathsf{SR}}(f) \qquad L_{\mathsf{SR}}(e \parallel f) = L_{\mathsf{SR}}(e) \parallel L_{\mathsf{SR}}(f)$$

A straightforward inductive argument now helps us validate the following:

Lemma 7.30. *For all $e \in \mathcal{T}$, we have $L_{\mathsf{SR}}(e) = [\![e]\!]$.*

7.2.2 Correctness of the syntactic PA

It remains to show that the syntactic PA satisfies the other objectives that we set, namely that it is fork-acyclic and bounded. For fork-acyclicity, we need to argue that if $f \in \mathcal{T}$ is a fork target of $e \in \mathcal{T}$, then $f \prec_{\mathsf{SR}} e$. To this end, we relate the support relation of the syntactic PA to $\parallel$-depth.

Lemma 7.31. *If $e, f \in \mathcal{T}$ such that $e \preceq_{\mathsf{SR}} f$, then $\mathsf{d}_{\parallel}(e) \leq \mathsf{d}_{\parallel}(f)$.*

To argue fork-acyclicity, it then suffices to show that if f is a fork target of e, it holds that the $\parallel$-depth of f is strictly lower than that of e, ensuring that e cannot support f.

Lemma 7.32. *Let $e, f \in \mathcal{T}$. If f is a fork target of e in the syntactic PA, then $\mathsf{d}_{\parallel}(f) < \mathsf{d}_{\parallel}(e)$. Consequently, the syntactic PA is fork-acyclic.*

Next, we prove that each state in the syntactic PA has finite support. To argue this, it suffices to find a finite overestimation of the support, i.e., a finite set and support-closed that contains that state; since the support of a state is the smallest such set, it is necessarily contained in such an overestimation. To that end, we propose the following:

Definition 7.33. We define $R : \mathcal{T} \to 2^{\mathcal{T}}$ inductively, as follows

$$R(0) = \{0\} \qquad R(e_1 + e_2) = R(e_1) \cup R(e_2) \cup \{e\} \qquad R(e_1^*) = R(e_1) \fatsemi e_1^* \cup R(e_1) \cup \{e_1^*\}$$
$$R(1) = \{1\} \qquad R(e_1 \cdot e_2) = R(e_1) \fatsemi e_2 \cup R(e_1) \cup R(e_2)$$
$$R(\mathtt{a}) = \{\mathtt{a}, 1\} \qquad R(e_1 \parallel e_2) = R(e_1) \cup R(e_2) \cup \{e_1 \parallel e_2, 1\}$$

It should be clear that $R(e)$ is finite; it also fulfills our objective, as follows.

Lemma 7.34. *For every $e \in \mathcal{T}$, we have that $e \in R(e)$ and $R(e)$ is support-closed.*

Consequently, the syntactic PA is bounded.

This then allows us to wrap up this section by stating half of our Kleene theorem, as follows.

Theorem 7.35 (Expressions to automata)**.** *For every $e \in \mathcal{T}$, we can obtain a fork-acyclic and finite PA A with a state q such that $L_A(q) = [\![e]\!]$.*

7.3 Automata to expressions

In this section, we provide the converse to the construction from the previous section, that is, we show that the language accepted by a state in any fork-acyclic and finite pomset automaton can be implemented by a series-rational expression. To achieve this, we start by deriving the conditions that these expressions should satisfy based on $\equiv$, the axiomatic equivalence relation on sr-expressions, and then proceed to derive the correct expressions from these conditions.

To develop an intuition of what the conditions on an expression that describes the language accepted by a state might look like, let $A = \langle Q, F, \delta, \gamma \rangle$ be a pomset automaton with $q \in Q$. If $q \in F$, then q accepts the empty pomset; hence, $1 \in L_A(q)$. Furthermore, let $\mathtt{a} \in \Sigma$ and $q' \in \delta(q, \mathtt{a})$. If we take a pomset in $L_A(q')$ and prepend $\mathtt{a}$, we should obtain a pomset in $L_A(q)$. Hence, we expect that $\mathtt{a} \cdot L_A(q') \subseteq L_A(q)$. Lastly, let $r_1, \dots, r_n \in Q$ and $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$. If for $1 \leq i \leq n$ we have $U_i \in L_A(r_i)$, and furthermore $V \in L_A(q')$, then $(U_1 \parallel \cdots \parallel U_n) \cdot V \in L_A(q)$. Hence, we should have that $(L_A(r_1) \parallel \cdots \parallel L_A(r_n)) \cdot L_A(q') \subseteq L_A(q)$. This leads us to the following characterisation.

Lemma 7.36. *If $A = \langle Q, F, \delta, \gamma \rangle$ be a pomset automaton, then $L_A : Q \to 2^{\mathsf{SP}}$ is the least function $t : Q \to 2^{\mathsf{SP}}$ (w.r.t. the pointwise inclusion order) such that for all $q \in Q$ the following hold:*

$$\frac{q \in F}{1 \in t(q)} \qquad \frac{\mathtt{a} \in \Sigma \qquad q' \in \delta(q, \mathtt{a})}{\mathtt{a} \cdot t(q') \subseteq t(q)} \qquad \frac{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})}{(t(r_1) \parallel \cdots \parallel t(r_n)) \cdot t(q') \subseteq t(q)}$$

We can exploit the above characterisation by taking it as a template for the conditions that we put on sr-expressions. This brings us to the idea of a *solution* to a pomset automaton, as follows.

Definition 7.37 (Solution of a PA)**.** Let $A = \langle Q, F, \delta, \gamma \rangle$ be a PA, and let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$. We say $s : Q \to \mathcal{T}(\Delta)$ is a $\approx$-*solution* to A if, for every $q \in Q$:

$$[q \in F] + \sum_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \lessapprox s(q)$$

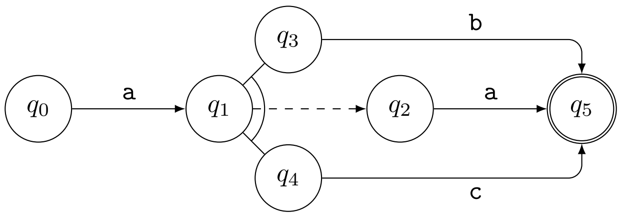


Figure 7.1: Pomset automaton accepting $\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c}) \cdot \mathtt{a}$. (repeated from page 139)

Also, s is a *least* $\approx$*-solution* to A if for every $\approx$-solution s' we have that $s(q) \lessapprox s(q')$ for all $q \in Q$. We call $s : Q \to \mathcal{T}$ the *least solution* to A if it is the least $\approx$-solution for any BKA congruence $\approx$.

Example 7.38. Let $A = \langle Q, F, \delta, \gamma \rangle$ be the PA in Figure 7.1, and let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$. The constraints on a $\approx$-solution $s : Q \to \mathcal{T}(\Delta)$ to A can then be written as

$$\mathtt{a} \cdot s(q_1) \lessapprox s(q_0) \qquad (s(q_3) \parallel s(q_4)) \cdot s(q_2) \lessapprox s(q_1) \qquad \mathtt{a} \cdot s(q_5) \lessapprox s(q_2)$$

$$\mathtt{b} \cdot s(q_5) \lessapprox s(q_3) \qquad \mathtt{c} \cdot s(q_5) \lessapprox s(q_4) \qquad 1 \lessapprox s(q_5)$$

Least $\approx$-solutions (resp. least solutions) to a PA are unique up to pointwise $\approx$-equivalence (resp. $\equiv$-equivalence). We therefore speak of *the* least $\approx$-solution or least solution, if it exists.

The least solution to a pomset automaton indeed contains the series-rational expressions that reflect the behaviour of its states, as validated by the following lemma.

Lemma 7.39. *Let $A = \langle Q, F, \delta, \gamma \rangle$ be a pomset automaton. If $s : Q \to \mathcal{T}$ is the least solution to A, then for $q \in Q$ it holds that $L_A(q) = [\![s(q)]\!]$.*

Proof. Recall the congruence $\doteq$ on $\mathcal{T}(2^{\mathsf{SP}})$, where $e \doteq f$ if and only if e and f represent the same sr-expression. In particular, Lemma 7.36 tells us that for $q \in Q$ we have

$$[q \in F] + \sum_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot L_A(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} (L_A(r_1) \parallel \cdots \parallel L_A(r_n)) \cdot L_A(q') \mathrel{\dot{\leq}} L_A(q)$$

where $\dot{\leq}$ is the precongruence associated with $\doteq$. Since L_A is also a function from Q to $\mathcal{T}(2^{\mathsf{SP}})$, and s is the least such function (w.r.t. $\dot{\leq}$), we have $s(q) \mathrel{\dot{\leq}} L_A(q)$, or equivalently, $[\![s(q)]\!] \subseteq L_A(q)$.

For the other direction, note that since s is a $\equiv$-solution, and $\equiv$ is sound w.r.t. $[\![-]\!]$, we have

$$\{1 \;:\; q \in F\} \cup \bigcup_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot [\![s(q')]\!] \cup \bigcup_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} ([\![s(r_1)]\!] \parallel \cdots \parallel [\![s(r_n)]\!]) \cdot [\![s(q')]\!] \subseteq [\![s(q)]\!]$$

Hence $[\![-]\!] \circ s$ satisfies the inference rules in Lemma 7.36; it follows that $L_A(q) \subseteq [\![s(q)]\!]$. □

We can therefore reach our objective if we just manage to find a method that obtains the least solution to any fork-acyclic and finite pomset automaton. To this end, we can leverage the results that we obtained about solving systems of series-rational expressions.

Lemma 7.40. *Let A be a fork-acyclic and finite PA. We can construct the least solution to A.*

Proof. We proceed by induction on the depth of $A = \langle Q, F, \delta, \gamma \rangle$. In the base, where $D_A = 0$, there cannot be any states, and therefore the claim holds vacuously. For the inductive step, assume that the claim holds for fork-acyclic and finite pomset automata of depth $D_A - 1$. We can then choose $Q' = \{q' \in Q \ :\ D_A(q) < D_A\}$, and we note that Q' is support-closed by construction. After all, if $q \in Q'$ and $q' \preceq_A q$, then $D_A(q') \leq D_A(q)$, and therefore $D_A(q') < D_A$, whence $q' \in Q'$.

This gives us $A' = A[Q'] = \langle Q', F \cap Q', \delta', \gamma' \rangle$; it is not hard to see that $D_{A'} < D_A$, and thus the induction hypothesis applies to A'. We then obtain $s' : Q' \to \mathcal{T}$ as the least solution to A'. Using s', we craft the series-rational system $\mathcal{S} = \langle M, b \rangle$ on $Q \setminus Q'$, where M and b are given by

$$M(q, q') = \sum_{q' \in \delta(q, \mathtt{a}) \setminus Q'} \mathtt{a} + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\}) \setminus Q'} s'(r_1) \parallel \cdots \parallel s'(r_n)$$

$$b(q) = [q \in F] + \sum_{q' \in \delta(q, \mathtt{a}) \cap Q'} \mathtt{a} \cdot s'(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\}) \cap Q'} (s'(r_1) \parallel \cdots \parallel s'(r_n)) \cdot s'(q')$$

In the above, we note that if r_i is a fork target of $q \in Q \setminus Q'$, then $r_i \prec_A q$ since A is fork-acyclic, and hence $D_A(r_i) < D_A(q) \leq D_A$, which means that $r_i \in Q'$; thus, M is well-defined. We can similarly argue that the calls to s' in the definition of $b(q)$ are well-defined.

Let $s'' : Q \setminus Q' \to \mathcal{T}$ be the least solution to $\mathcal{S}$. We choose $s : Q \to \mathcal{T}$ by setting $s(q) = s'(q)$ when $q \in Q'$, and $s(q) = s''(q)$ when $q \in Q \setminus Q'$. We claim that s is the least solution to A. To this end, let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$, with associated precongruence $\lessapprox$. To see that it is a $\approx$-solution at all, first note that if $q \in Q'$, then we can derive that

$$\begin{aligned}
&[q \in F] + \sum_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\
&= [q \in F] + \sum_{q' \in \delta'(q, \mathtt{a})} \mathtt{a} \cdot s'(q') + \sum_{q' \in \gamma'(q, \{\!|r_1, \dots, r_n|\!\})} (s'(r_1) \parallel \cdots \parallel s'(r_n)) \cdot s'(q') && (\text{def. } s, \delta', \gamma') \\
&\lessapprox s'(q) = s(q) && (s' \text{ is a } \approx\text{-solution to } A')
\end{aligned}$$

Otherwise, if $q \in Q \setminus Q'$, then we calculate that

$$[q \in F] + \sum_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q')$$

$$\begin{aligned}
&\approx [q \in F] + \sum_{q' \in \delta(q,\mathtt{a}) \cap Q'} \mathtt{a} \cdot s'(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\}) \cap Q'} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s'(q') \\
&\qquad + \sum_{q' \in \delta(q,\mathtt{a}) \setminus Q'} \mathtt{a} \cdot s''(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\}) \setminus Q'} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s''(q') && \text{(def. } s\text{)} \\
&\approx b(q) + \sum_{q' \in \delta(q,\mathtt{a}) \setminus Q'} \mathtt{a} \cdot s''(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\}) \setminus Q'} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s''(q') && \text{(def. } b\text{)} \\
&\lessapprox s''(q) = s(q) && (s'' \text{ is a } \langle \approx, 1 \rangle\text{-solution to } \mathcal{S})
\end{aligned}$$

To see that s is the *least* $\approx$-solution, suppose that $t : Q \to \mathcal{T}(\Delta)$ is a $\approx$-solution to A. If t' is the restriction of t to Q', then t' becomes a $\approx$-solution to A', and thus we find for $q \in Q'$ that $s(q) = s'(q) \lessapprox t'(q) = t(q)$. Furthermore, if t'' is the restriction of t to $Q \setminus Q'$, then t'' becomes a $\approx$-solution to $\mathcal{S}$, and thus we find for $q \in Q \setminus Q'$ that $s(q) = s''(q) \lessapprox t''(q) = t(q)$. $\square$

This allows us to conclude with the converse of Theorem 7.35. Together, these theorems combine into a Kleene theorem for pomset languages, which was the goal of this chapter.

Theorem 7.41. *If $A = \langle Q, F, \delta, \gamma \rangle$ is a fork-acyclic and finite PA, then we can construct for every $q \in Q$ a series-rational expression $e \in \mathcal{T}$ such that $L_A(q) = [\![e]\!]$.*

Corollary 7.42 (Kleene theorem for pomset languages)**.** *Let $L \subseteq \mathsf{SP}$. Then L is series-rational if and only if it is accepted by a finite and fork-acyclic pomset automaton.*

Summary of this chapter We introduced pomset automata as an operational formalism to accept series-parallel pomset languages. Using Antimirov's construction, we showed that every pomset language expressed by a series-rational expression can also be accepted by a finite and fork-acyclic pomset automaton. Conversely, we used series-rational systems to argue that a pomset language accepted by a fork-acyclic and finite pomset automaton can also be represented by a series-rational expression, thereby establishing an extension of Kleene's celebrated theorem.

7.A Proofs about pomset automata

Lemma 7.8. *Let $q \xrightarrow{U}_A q'$. There exist $q = q_0, \ldots, q_\ell = q' \in Q$ and $U_1, \ldots, U_\ell \in \mathsf{SP}$, such that $U = U_1 \cdots U_\ell$, and for all $1 \leq i \leq \ell$ we have that $q_{i-1} \xrightarrow{U_i}_A q_i$ is a unit run.*

Proof. We proceed by induction on $\to_A$. In the base, there are two cases to consider.

- If $q \xrightarrow{U}_A q'$ because $q = q'$ and $U = 1$, then we choose $\ell = 0$ to satisfy the claim.
- If $q \xrightarrow{U}_A q'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q' \in \delta(q, \mathtt{a})$, then choose $\ell = 1$ and $U_1 = \mathtt{a}$.

For the inductive step, there are also two cases to consider.

- On the one hand, if $q \xrightarrow{U}_A q'$ because $U = V \cdot W$ and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$, then we can obtain the necessary states and pomsets by induction. More precisely, let $q = q''_0, \dots, q''_{\ell''} = q'' \in Q$ and $V_1, \dots, V_{\ell''} \in \mathsf{SP}$ as well as $q'' = q'_0, \dots, q'_{\ell'} = q' \in Q$ and $W_1, \dots, W_{\ell'} \in \mathsf{SP}$ such that $V = V_1 \cdots V_{\ell''}$ and $W = W_1 \cdots W_{\ell'}$ and for all $1 \leq i \leq \ell''$ we have that $q''_{i-1} \xrightarrow{V_i}_A q''_i$ is a unit run, while for all $1 \leq i \leq \ell'$ we have that $q'_{i-1} \xrightarrow{W_i}_A q'_i$ is a unit run. We can then choose $\ell = \ell'' + \ell'$, and $U_1, \dots, U_\ell$ such that for $1 \leq i \leq \ell''$ we have $U_i = V_i$, and for $\ell'' < i \leq \ell$ we have $U_i = W_{i-\ell''}$, as well as $q = q_0, \dots, q_\ell = q' \in Q$ such that for $0 \leq i \leq \ell''$ we have $q_i = q''_i$, and for $\ell'' < i \leq \ell$ we have $q_i = q'_{i-\ell''}$. It is then straightforward to see that $U = V \cdot W = V_1 \cdots V_{\ell''} \cdot W_1 \cdots W_{\ell'} = U_1 \cdots U_{\ell''} \cdot U_{\ell''+1} \cdots U_\ell$, and that for $1 \leq i \leq n$ we have that $q_{i-1} \xrightarrow{U_i}_A q_i$ is a unit run.
- On the other hand, if $q \xrightarrow{U}_A q'$ because $U = U_1 \parallel \cdots \parallel U_n$ and there exist $r_1, \dots, r_n \in Q$ as well as $r'_1, \dots, r'_n \in F$ such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U'_i} r'_i$, and furthermore $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, then we can choose $\ell = 1$ and $U_1 = U$ to satisfy the claim. □

Lemma 7.20. *If Q' is support-closed, then $A[Q']$ implements A. If A is bounded, then so is $A[Q']$.*

Proof. We should show that, for $q \in Q'$, it holds that $L_{A[Q']}(q) = L_A(q)$. For the inclusion from left to right, we prove that if $q \xrightarrow{U}_{A[Q']} q'$, then $q \xrightarrow{U}_A q'$. We proceed by induction on $\rightarrow_{A[Q']}$. In the base, there are two cases. On the one hand, if $U = 1$ and $q = q'$, then $q \xrightarrow{U}_A q'$ immediately. Otherwise, if $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q' \in \delta'(q, \mathtt{a})$, then $q' \in \delta(q, \mathtt{a})$, and hence $q \xrightarrow{U}_A q'$ as well.

For the inductive step, there are two cases to consider.

- Suppose that $q \xrightarrow{U}_{A[Q']} q'$ because $U = V \cdot W$ and there exists a $q'' \in Q'$ such that $q \xrightarrow{V}_{A[Q']} q''$ and $q'' \xrightarrow{W}_{A[Q']} q'$. By induction, $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$, and therefore $q \xrightarrow{U}_A q'$.
- Suppose that $q \xrightarrow{U}_{A[Q']} q'$ because $U = U_1 \parallel \cdots \parallel U_n$ and $r_1, \dots, r_n \in Q$ and $r'_1, \dots, r'_n \in Q' \cap F$ s.t. for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_{A[Q']} r'_i$ as well as $q' \in \gamma'(q, \{\!|r_1, \dots, r_n|\!\})$. By induction, we find for $1 \leq i \leq n$ that $r_i \xrightarrow{U_i}_A r'_i \in F$. Since $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, we conclude $q \xrightarrow{U}_A q'$.

For the other inclusion, we prove that if $q \in Q'$ and $q \xrightarrow{U}_A q'$ then $q' \in Q'$ and $q \xrightarrow{U}_{A[Q']} q'$. The proof proceeds by induction on $\rightarrow_A$. In the base, there are two cases.

- If $U = 1$ and $q = q'$, then $q' \in Q'$ and $q \xrightarrow{U}_{A[Q']} q'$ immediately.
- If $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $q' \in \delta(r, \mathtt{a})$, then $q' \preceq_A q$, and hence $q' \in Q'$. It then follows that $q' \in \delta'(q, \mathtt{a})$, which allows us to conclude that $q \xrightarrow{U}_{A[Q']} q'$.

For the inductive step, there are two cases to consider.

- Suppose that $q \xrightarrow{U}_A q'$ because $U = V \cdot W$, and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$. It follows that $q'' \in Q'$ and $q \xrightarrow{V}_{A[Q']} q''$ by induction. Similarly, we find $q' \in Q'$ and $q'' \xrightarrow{W}_{A[Q']} q'$, again by induction. We then conclude that $q \xrightarrow{U}_{A[Q']} q'$.
- Suppose that $q \xrightarrow{U}_A q'$ because $U = U_1 \parallel \cdots \parallel U_n$, and $r_1, \dots, r_n \in Q$ and $q'_1, \dots, q'_n \in F'$ such that for $1 \leq i \leq n$ we have $q_i \xrightarrow{U_i}_A q'_i$, as well as $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$. When have $r_1, \dots, r_n \preceq_A q$, and thus $r_1, \dots, r_n \in Q'$, as well as $q' \in Q'$. By induction, we then find for $1 \leq i \leq n$ that $r'_i \in Q'$ and $r_i \xrightarrow{U_i}_{A[Q']} r'_i$. Since $q' \in \gamma'(q, \{\!|r_1, \dots, r_n|\!\})$, also $q \xrightarrow{U}_{A[Q']} q'$.

Lastly, we note that for $q, q' \in Q$, we have $q \preceq_{A[Q']} q'$ if and only if $q \preceq_A q'$ and $q, q' \in Q'$; hence, if A is fork-acyclic, then so is $A[Q']$. Furthermore, if $q \in Q'$, then the support of q in $A[Q']$ is the same as the support of q in A; hence, if A is bounded, then $A[Q']$ must also be bounded. □

Lemma 7.36. *If $A = \langle Q, F, \delta, \gamma \rangle$ be a pomset automaton, then $L_A : Q \to 2^{\mathsf{SP}}$ is the least function $t : Q \to 2^{\mathsf{SP}}$ (w.r.t. the pointwise inclusion order) such that for all $q \in Q$ the following hold:*

$$\frac{q \in F}{1 \in t(q)} \qquad \frac{\mathtt{a} \in \Sigma \qquad q' \in \delta(q, \mathtt{a})}{\mathtt{a} \cdot t(q') \subseteq t(q)} \qquad \frac{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})}{(t(r_1) \parallel \cdots \parallel t(r_n)) \cdot t(q') \subseteq t(q)}$$

Proof. Let t be the least function satisfying the rules above. To show that for all $q \in Q$ we have $t(q) \subseteq L_A(q)$, it suffices to show that L_A satisfies the same property as t. There are three cases.

- First, suppose that $q \in F$. In that case, $q \xrightarrow{1}_A q \in F$, whence $1 \in L_A(q)$ immediately.
- Next, suppose that $\mathtt{a} \in \Sigma$ and $q' \in \delta(q, \mathtt{a})$, and let $U \in \mathtt{a} \cdot L_A(q')$. In that case, $U = \mathtt{a} \cdot U'$ for some $U' \in L_A(q')$. Hence, there exists a $q'' \in F$ such that $q' \xrightarrow{U'}_A q''$; since $q' \in \delta(q, \mathtt{a})$ we also have that $q \xrightarrow{\mathtt{a}}_A q'$, and therefore $q \xrightarrow{U}_A q'' \in F$, which means that $U \in L_A(q)$.
- Lastly, suppose that $r_1, \dots, r_n \in Q$ and $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, and let $U \in (L_A(r_1) \parallel \cdots \parallel L_A(r_n)) \cdot L_A(q')$. In that case, $U = (V_1 \parallel \cdots \parallel V_n) \cdot W$ such that for $1 \leq i \leq n$ we have $V_i \in L_A(r_i)$, and furthermore $W \in L_A(q')$. This means that for $1 \leq i \leq n$ we have $r'_i \in F$ such that $r_i \xrightarrow{V_i}_A r'_i$ and also $q'' \in F$ such that $q' \xrightarrow{W}_A q''$. Since $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$ we have that $q \xrightarrow{V_1 \parallel \cdots \parallel V_n}_A q'$. In total, we find that $q \xrightarrow{U}_A q''$, and thus $U \in L_A(q)$.

For the converse, we show that if $q \xrightarrow{U}_A q'$ and $V \in t(q')$, then $U \cdot V \in t(q)$, by induction on $q \xrightarrow{U}_A q'$. In the base, there are two cases to consider.

- If $q \xrightarrow{U}_A q'$ because $q = q'$ and $U = 1$, then $U \cdot V = V \in t(q') = t(q)$.

- If $q \xrightarrow{U}_A q'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ with $q' \in \delta(q, \mathtt{a})$, then $U \cdot V \in \mathtt{a} \cdot t(q') \subseteq t(q)$.

For the inductive step, there are again two cases to consider.

- Suppose that $q \xrightarrow{U}_A q'$ because $U = W \cdot X$ and there exists a $q'' \in Q$ such that $q \xrightarrow{W}_A q''$ and $q'' \xrightarrow{X}_A q'$. Applying induction twice, we find that $U \cdot V = W \cdot X \cdot V \subseteq W \cdot t(q'') \subseteq t(q)$.
- Suppose that $q \xrightarrow{U}_A q'$ because $U = U_1 \parallel \cdots \parallel U_n$ and there exist $r_1, \dots, r_n \in Q$ as well as $r'_1, \dots, r'_n \in F$ such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r'_i$, and $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$. For $1 \leq i \leq n$ we have that $1 \in L_A(r'_i)$, and hence by induction it follows that $U_i = U_i \cdot 1 \in t(r_i)$. We can then conclude that $U \cdot V = (U_1 \parallel \cdots \parallel U_n) \cdot V \in (t(r_1) \parallel \cdots \parallel t(r_n)) \cdot t(q') \subseteq t(q)$.

Concluding our proof of the first claim, we note that if $U \in L_A(q)$, then there exists a $q' \in F$ such that $q \xrightarrow{U}_A q'$. Since $1 \in t(q')$ by definition, it follows that $U = U \cdot 1 \in t(q)$. □

7.B Proofs about the syntactic pomset automaton

Lemma 7.25. *Let $e_1, e_2 \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists an $f \in \mathcal{F}$ such that $e_1 + e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) There exists an $f \in \mathcal{F}$ such that $e_1 \xrightarrow{U}_{\mathsf{SR}} f$ or $e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

Proof. To show that (i) implies (ii), there are two cases, depending on the length ℓ of $e_1 + e_2 \xrightarrow{U}_{\mathsf{SR}} f$:

- If $\ell = 0$, then $e_1 + e_2 \xrightarrow{U}_{\mathsf{SR}} f$ is trivial. In that case, $U = 1$ and $f = e_1 + e_2$, and so $e_i \in \mathcal{F}$ for some $i \in \{0, 1\}$; if we choose $f' = e_i$, we find $e_i \xrightarrow{U}_{\mathsf{SR}} f'$, and the claim is satisfied.
- Otherwise, if $\ell > 0$, then $U = U_0 \cdot U'$ and there exists a $g \in \mathcal{T}$ such that $e_1 + e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a unit run, and $g \xrightarrow{U'}_{\mathsf{SR}} f$ is of length ℓ.

  If $e_1 + e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $g \in \delta_{\mathsf{SR}}(e_1 + e_2, \mathtt{a})$. Without loss of generality, let $g \in \delta_{\mathsf{SR}}(e_1, \mathtt{a})$; in that case, $e_1 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a unit run. If we then choose $f' = f$, we find that $e_1 \xrightarrow{U}_{\mathsf{SR}} f'$ is of length ℓ. The case where $e_1 + e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a parallel unit run is similar.

To show that (ii) implies (i), we treat the case where $e_1 \xrightarrow{U}_{\mathsf{SR}} f$; the case where $e_2 \xrightarrow{U}_{\mathsf{SR}} f$ is similar. There are again two cases, depending on the length ℓ of $e_1 \xrightarrow{U}_{\mathsf{SR}} f_1$.

- If $\ell = 0$, then $f_1 = e_1$ and $U = 1$. We then choose $f' = e_1 + e_2$.

- If $\ell > 0$, then we find $e_1' \in \mathcal{T}$ and $U = U_0 \cdot U'$ such that $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a unit run, and $e_1' \xrightarrow{U'}_{\mathsf{SR}} f_1$. If $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $e_1' \in \delta_{\mathsf{SR}}(e_1, \mathtt{a})$. But then $e_1' \in \delta_{\mathsf{SR}}(e_1 + e_2, \mathtt{a})$ as well, and hence $e_1 + e_2 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$. Putting this together, we find that $e_1 + e_2 \xrightarrow{U}_{\mathsf{SR}} f_1$; choosing $f' = f_1$ then satisfies the claim.

  The case where $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a parallel unit run is similar. □

Lemma 7.26. *Let $e_1, e_2 \in \mathcal{T}$, $U \in \mathsf{SP}$, and $\ell \in \mathbb{N}$. The following are equivalent:*

(i) There exists an $f \in \mathcal{F}$ such that $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f$ of length ℓ.

(ii) $U = U_1 \cdot U_2$, and there exist $f_1, f_2 \in \mathcal{F}$ with $e_i \xrightarrow{U_i}_{\mathsf{SR}} f_i$ for $i \in \{1, 2\}$, of total length at most ℓ.

Proof. To show that (i) implies (ii), we proceed by induction on the length ℓ of $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f$. In the base, where $\ell = 0$, we have $f = e_1 \cdot e_2$ (hence $e_1, e_1 \in \mathcal{F}$) and $U = 1$. We can then choose $f_1 = e_1$ and $f_2 = e_2$ as well as $U_1 = U_2 = 1$, to find that $e_1 \xrightarrow{U_1}_{\mathsf{SR}} f_1$ and $e_2 \xrightarrow{U_2}_{\mathsf{SR}} f_1$, of length zero.

For the inductive step, let $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f$ be of length $\ell + 1$. We find $U = U_0 \cdot U'$, and $g \in \mathcal{T}$ where $e_1 \cdot e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a unit run, and $g \xrightarrow{U'}_{\mathsf{SR}} f$ is of length ℓ. If $e_1 \cdot e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $g \in \delta_{\mathsf{SR}}(e_1 \cdot e_2, \mathtt{a})$. This gives us two cases.

- If $g \in \delta_{\mathsf{SR}}(e_1, \mathtt{a}) \fatsemi e_2$, then $g = g_1 \cdot e_2$ such that $g_1 \in \delta_{\mathsf{SR}}(e_1, \mathtt{a})$. By induction we find $f_1, f_2 \in \mathcal{F}$ and $U' = U_1' \cdot U_2'$ such that $g_1 \xrightarrow{U_1'}_{\mathsf{SR}} f_1$, and $e_2 \xrightarrow{U_2'}_{\mathsf{SR}} f_2$, of total length at most ℓ. We choose $U_1 = U_0 \cdot U_1'$ and $U_2 = U_2'$ to find $U = U_0 \cdot U' = U_0 \cdot U_1' \cdot U_2' = U_1 \cdot U_2$, such that $e_1 \xrightarrow{U_1}_{\mathsf{SR}} f_1$ and $e_2 \xrightarrow{U_2}_{\mathsf{SR}} f_2$ of total length at most $\ell + 1$.

- If $g \in e_1 \star \delta_{\mathsf{SR}}(e_2, \mathtt{a})$, then first note that $e_1 \in \mathcal{F}$, and $g \in \delta_{\mathsf{SR}}(e_2, \mathtt{a})$. We choose $U_1 = 1$ and $U_2 = U$ as well as $f_1 = e_1$ and $f_2 = f'$ to find that $U = U_1 \cdot U_2$ as well as $e_1 \xrightarrow{U_1}_{\mathsf{SR}} f_1$ of length zero. Lastly, $e_2 \xrightarrow{U_0}_{\mathsf{SR}} g \xrightarrow{U'}_{\mathsf{SR}} f' = f_2$, meaning $e_2 \xrightarrow{U}_{\mathsf{SR}} f_2$ of length at most $\ell + 1$.

The case where $e_1 \cdot e_2 \xrightarrow{U_0}_{\mathsf{SR}} g$ is a parallel unit run can be treated similarly.

The proof that (ii) implies (i) consists of two phases; first, we verify the following.

Fact 7.B.1. *We have that $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f_1 \cdot e_2$, of the same length as $e_1 \xrightarrow{U}_{\mathsf{SR}} f_1$.*

Proof of Fact 7.B.1. The proof proceeds by induction on the length ℓ of $e_1 \xrightarrow{U}_{\mathsf{SR}} f_1$. In the base, where $\ell = 0$ and $f_1 = e_1$ as well as $U = 1$, we the claim holds immediately.

In the inductive step, let $e_1 \xrightarrow{U}_{\mathsf{SR}} f_1$ be of length $\ell + 1$. We find $e_1' \in \mathcal{T}$ and $U = U_0 \cdot U'$ such that $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a unit run, and $e_1' \xrightarrow{U'}_{\mathsf{SR}} f_1$ is of length ℓ. By induction, $e_1' \cdot e_2 \xrightarrow{U'}_{\mathsf{SR}} f_1 \cdot e_2$. If $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $e_1' \in \delta_{\mathsf{SR}}(e_1, \mathtt{a})$, meaning $e_1' \cdot e_2 \in \delta_{\mathsf{SR}}(e_1 \cdot e_2, \mathtt{a})$, hence $e_1 \cdot e_2 \xrightarrow{U_0}_{\mathsf{SR}} e_1' \cdot e_2$. We conclude that $e_1 \cdot e_2 \xrightarrow{U}_{\mathsf{SR}} f_1 \cdot e_2$.

The case where $e_1 \xrightarrow{U_0}_{\mathsf{SR}} e_1'$ is a parallel unit run is similar. □

Next, we note the following.

Fact 7.B.2. *There exists an $f \in \mathcal{F}$ such that $f_1 \cdot e_2 \xrightarrow{V}_{\mathsf{SR}} f$, of the same length as $e_2 \xrightarrow{V}_{\mathsf{SR}} f_2$.*

Proof of Fact 7.B.2. There are two cases to consider, based on the length of $e_2 \xrightarrow{V}_{\mathsf{SR}} f_1$.

- If $\ell = 0$, then we know that $f_2 = e_2$ and $V = 1$. We choose $f = f_1 \cdot e_2$.
- In the inductive step, let $e_2 \xrightarrow{V}_{\mathsf{SR}} f_1$ be of length $\ell + 1$. We find $e_2' \in \mathcal{T}$ and $V = V_0 \cdot V'$ such that $e_2 \xrightarrow{V_0}_{\mathsf{SR}} e_2'$ is a unit run, and $e_2' \xrightarrow{V'}_{\mathsf{SR}} f_2$ is of length ℓ. If $e_2 \xrightarrow{V_0}_{\mathsf{SR}} e_2'$ is a sequential unit run, then $V_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $e_2' \in \delta_{\mathsf{SR}}(e_2, \mathtt{a})$, and thus $e_2' \in \delta_{\mathsf{SR}}(f_1 \cdot e_2, \mathtt{a})$. Hence, $f_1 \cdot e_2 \xrightarrow{V_0}_{\mathsf{SR}} e_2'$, meaning $f_1 \cdot e_2 \xrightarrow{V}_{\mathsf{SR}} f_2$; choosing $f = f_2$ satisfies the claim.

  The case where $e_2 \xrightarrow{V_0}_{\mathsf{SR}} e_2'$ is a parallel unit run is similar. □

Putting Facts 7.B.1 and 7.B.2 together, we find $f \in \mathcal{F}$ such that $e_1 \cdot e_2 \xrightarrow{U \cdot V}_{\mathsf{SR}} f$. □

Lemma 7.27. *Let $e_1, e_2 \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e_1 \parallel e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) $U = U_1 \parallel U_2$ and there exist $f_1, f_2 \in \mathcal{F}$ such that $e_i \xrightarrow{U_i}_{\mathsf{SR}} f_i$ for $i \in \{1, 2\}$.

Proof. To show that (i) implies (ii), there are two cases, based on the length ℓ of $e_1 \parallel e_2 \xrightarrow{U}_{\mathsf{SR}} f$.

- If $\ell = 1$, then $e_1 \parallel e_2 \xrightarrow{U}_{\mathsf{SR}} f$ is trivial, then $U = 1$ and $e_1 \parallel e_2 = f \in \mathcal{F}$. Hence, $e_1, e_2 \in \mathcal{F}$; we can choose $f_1 = e_1$, $f_2 = e_2$ and $U_1 = U_2 = 1$ to satisfy the claim.
- Otherwise, if $\ell > 0$, then there there must exist $U_0, U' \in \mathsf{SP}$ and $g \in \mathcal{T}$ such that $U = V \cdot W$ and $e_1 \parallel e_2 \xrightarrow{V}_{\mathsf{SR}} g$ is a unit run, and $g \xrightarrow{W}_{\mathsf{SR}} f$.

  We can discount the possibility that $e_1 \parallel \xrightarrow{V}_{\mathsf{SR}} g$ is a sequential unit run, because $\delta_{\mathsf{SR}}(e_1 \parallel e_2, \mathtt{a}) = \emptyset$ for all $\mathtt{a} \in \Sigma$. Hence, $e_1 \parallel e_2 \xrightarrow{V}_{\mathsf{SR}} g$ is a parallel unit run, meaning that $V = V_1 \parallel \cdots \parallel V_n$ and there exists a $\phi = \{\!|h_1, \dots, h_n|\!\} \in \mathbb{M}(\mathcal{T})$ such that $g \in \gamma_{\mathsf{SR}}(e_1 \parallel e_1, \phi)$, and for $1 \leq i \leq n$ there exists an $h_i' \in \mathcal{F}$ with $h_i \xrightarrow{V_i}_{\mathsf{SR}} h_i'$. By definition of γ_{SR}, it then follows that $n = 2$ and $g = 1$ as well as (without loss of generality) $e_1 = h_1$ and $e_2 = h_2$. Since $g = 1$, it must be that $g \xrightarrow{W}_{\mathsf{SR}} f$ is trivial, and hence $W = 1$, meaning that $U = V$. We choose $f_1 = h_1'$, $f_2 = h_2'$, $U_1 = V_1$ and $U_2 = V_2$ to satisfy the claim.

The other direction, i.e., that (ii) implies (i), holds by construction of the syntactic PA. □

Lemma 7.28. *Let $e \in \mathcal{T}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e^* \xrightarrow{U}_{\mathsf{SR}} f$.

(ii) $U = U_1 \cdots U_n$, such that for $1 \leq i \leq n$ there exists $f_i \in \mathcal{F}$ with $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$.

Proof. The proof that (i) implies (ii) proceeds by induction on the length ℓ of $e^* \xrightarrow{U}_{\mathsf{SR}} f$. In the base, where $\ell = 0$, we have that $f = e^*$ and $U = 1$; it suffices to choose $n = 0$.

In the inductive step, let $e^* \xrightarrow{U}_{\mathsf{SR}} f$ be of length $\ell + 1$. We find $g \in \mathcal{T}$ and $U = U_0 \cdot U'$ such that $e^* \xrightarrow{U_0}_{\mathsf{SR}} g$ is a unit run, and $g \xrightarrow{U'}_{\mathsf{SR}} f$ is of length ℓ. If $e^* \xrightarrow{U_0}_{\mathsf{SR}} g$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $g \in \delta_{\mathsf{SR}}(e^*, \mathtt{a}) = \delta_{\mathsf{SR}}(e, \mathtt{a}) \fatsemi e^*$. Hence, $g = g' \cdot e^*$, with $g' \in \delta_{\mathsf{SR}}(e, \mathtt{a})$. By Lemma 7.26, we find $f'', f' \in \mathcal{F}$ such that $U' = V \cdot W$ as well as $g' \xrightarrow{V}_{\mathsf{SR}} f''$ and $e^* \xrightarrow{W}_{\mathsf{SR}} f'$, of length at most ℓ. By induction, we find $f_2, f_3, \dots, f_n \in \mathcal{F}$ such that $W = U_2 \cdot U_3 \cdots U_n$, and for $1 < i \leq n$ we have $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$. We choose $f_1 = f''$ and $U_1 = U_0 \cdot V$. For these choices, $U = U_0 \cdot U' = U_0' \cdot V \cdot W = U_1 \cdot U_2 \cdots U_n$. Since $e \xrightarrow{U_0}_{\mathsf{SR}} \delta_{\mathsf{SR}}(e, \mathtt{a}) \xrightarrow{V}_{\mathsf{SR}} f'$, we also have $e \xrightarrow{U_1}_{\mathsf{SR}} f_1$.

The case where $e^* \xrightarrow{U_0}_{\mathsf{SR}} g$ is a parallel unit run is similar.

To show that (ii) implies (i), we can assume w.l.o.g. that for $0 \leq i < n$ it holds that $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$ is non-trivial. We proceed by induction on n. In the base, where $n = 0$, we can choose $f = e^*$.

For the inductive step, assume that the claim holds for $n - 1$. By induction, we find $f' \in \mathcal{F}$ with $e^* \xrightarrow{U_2 \cdot U_3 \cdots U_n}_{\mathsf{SR}} f'$. Since $e \xrightarrow{U_1}_{\mathsf{SR}} f_1$ is non-trivial, we find $e' \in \mathcal{T}$ and $U_1 = U_0 \cdot U_1'$ with $e \xrightarrow{U_0}_{\mathsf{SR}} e'$ is a unit run, and $e' \xrightarrow{U_1'}_{\mathsf{SR}} f_1$. By Lemma 7.26, we find $f \in \mathcal{F}$ with $e' \cdot e^* \xrightarrow{U_1' \cdot U_2 \cdot U_3 \cdots U_n}_{\mathsf{SR}} f$. If $e \xrightarrow{U_0}_{\mathsf{SR}} e'$ is a sequential unit run, then $U_0 = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $e' \in \delta_{\mathsf{SR}}(e, \mathtt{a})$. But then $e' \cdot e^* \in \delta_{\mathsf{SR}}(e^*, \mathtt{a})$, and hence $e^* \xrightarrow{U_0}_{\mathsf{SR}} e' \cdot e^*$. Thus, $e^* \xrightarrow{U_0}_{\mathsf{SR}} e' \cdot e^* \xrightarrow{U_1' \cdot U_2 \cdot U_3 \cdots U_n}_{\mathsf{SR}} f$.

The case where $e \xrightarrow{U_0}_{\mathsf{SR}} e'$ is a parallel unit run is similar. □

Lemma 7.29. *Let $e, f \in \mathcal{T}$. The following hold:*

$$L_{\mathsf{SR}}(e + f) = L_{\mathsf{SR}}(e) + L_{\mathsf{SR}}(f) \qquad L_{\mathsf{SR}}(e^*) = L_{\mathsf{SR}}(e)^*$$

$$L_{\mathsf{SR}}(e \cdot f) = L_{\mathsf{SR}}(e) \cdot L_{\mathsf{SR}}(f) \qquad L_{\mathsf{SR}}(e \parallel f) = L_{\mathsf{SR}}(e) \parallel L_{\mathsf{SR}}(f)$$

Proof. We treat the claims case-by-case.

- To show $L_{\mathsf{SR}}(e + f) = L_{\mathsf{SR}}(e) + L_{\mathsf{SR}}(f)$, suppose $U \in L_{\mathsf{SR}}(e + f)$, i.e., $e + f \xrightarrow{U}_{\mathsf{SR}} g$ for $g \in \mathcal{F}$. By Lemma 7.25, we find $g' \in \mathcal{F}$ with $e \xrightarrow{U}_{\mathsf{SR}} g'$ or $f \xrightarrow{U}_{\mathsf{SR}} g'$, and hence $U \in L_{\mathsf{SR}}(e) + L_{\mathsf{SR}}(f)$.

  For the other inclusion, suppose that $U \in L_{\mathsf{SR}}(e)$. We then have that $e \xrightarrow{U}_{\mathsf{SR}} g$ for some $g \in \mathcal{F}$. By Lemma 7.25, there exists a $g' \in \mathcal{F}$ such that $e + f \xrightarrow{U}_{\mathsf{SR}} g'$, and hence $U \in L_{\mathsf{SR}}(e + f)$. The case where $U \in L_{\mathsf{SR}}(f)$ can be treated similarly.

- To show $L_{\mathsf{SR}}(e \cdot f) = L_{\mathsf{SR}}(e) \cdot L_{\mathsf{SR}}(f)$, suppose that $U \in L_{\mathsf{SR}}(e \cdot f)$, i.e., $e \cdot f \xrightarrow{U}_{\mathsf{SR}} g$ for some $g \in \mathcal{F}$. By Lemma 7.26, we find $g_0, g_1 \in \mathcal{F}$ such that $U = U_0 \cdot U_1$ as well as $e \xrightarrow{U_0}_{\mathsf{SR}} g_0$ and $f \xrightarrow{U_1}_{\mathsf{SR}} g_1$. This means that $U_0 \in L_{\mathsf{SR}}(e)$ and $U_1 \in L_{\mathsf{SR}}(f)$, and thus $U \in L_{\mathsf{SR}}(e) \cdot L_{\mathsf{SR}}(f)$.

  If $U \in L_{\mathsf{SR}}(e) \cdot L_{\mathsf{SR}}(f)$, then $U = U_0 \cdot U_1$ such that $U_0 \in L_{\mathsf{SR}}(e)$ and $U_1 \in L_{\mathsf{SR}}(f)$. This means that there exist $g_0, g_1 \in \mathcal{F}$ such that $e \xrightarrow{U_0}_{\mathsf{SR}} g_0$ and $f \xrightarrow{U_1}_{\mathsf{SR}} g_1$. By Lemma 7.26, there exists a $g \in \mathcal{F}$ such that $e \cdot f \xrightarrow{U}_{\mathsf{SR}} g$, and hence $U \in L_{\mathsf{SR}}(e \cdot f)$.

- To show $L_{\mathsf{SR}}(e \parallel f) = L_{\mathsf{SR}}(e) \parallel L_{\mathsf{SR}}(f)$, suppose $U \in L_{\mathsf{SR}}(e \parallel f)$, i.e., $e \parallel f \xrightarrow{U}_{\mathsf{SR}} g$ for some $g \in \mathcal{F}$. By Lemma 7.27, we find $g_1, g_2 \in \mathcal{F}$ and $U_1, U_2 \in \mathsf{SP}$ such that $U = U_1 \parallel U_2$ as well as $e_1 \xrightarrow{U_1}_{\mathsf{SR}} g_1$ and $e_2 \xrightarrow{U_2}_{\mathsf{SR}} g_2$. It then follows that $U = U_1 \parallel U_2 \in L_{\mathsf{SR}}(e) \parallel L_{\mathsf{SR}}(f)$.

  If $U \in L_{\mathsf{SR}}(e) \parallel L_{\mathsf{SR}}(f)$, then $U = U_1 \parallel U_2$ such that $U_1 \in L_{\mathsf{SR}}(e)$ and $U_2 \in L_{\mathsf{SR}}(f)$. This means that there exist $g_1, g_2 \in \mathcal{F}$ such that $e \xrightarrow{U_1}_{\mathsf{SR}} g_1$ and $f \xrightarrow{U_2}_{\mathsf{SR}} g_2$. By Lemma 7.27, we find that $e \parallel f \xrightarrow{U}_{\mathsf{SR}} 1 \in \mathcal{F}$, and hence $U \in L_{\mathsf{SR}}(e \parallel f)$.

- To show $L_{\mathsf{SR}}(e^*) = L_{\mathsf{SR}}(e)^*$, suppose $U \in L_{\mathsf{SR}}(e^*)$, i.e., $e^* \xrightarrow{U}_{\mathsf{SR}} f$ for $f \in \mathcal{F}$. By Lemma 7.28, we find that $U = U_1 \cdots U_n$ and $f_1, \dots, f_n \in \mathcal{F}$ such that for $1 \leq i \leq n$ we have $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$. Hence, we know for $1 \leq i \leq n$ that $U_i \in L_{\mathsf{SR}}(e)$, and therefore $U = U_1 \cdots U_n \in L_{\mathsf{SR}}(e)^*$.

  For the other direction, let $U \in L_{\mathsf{SR}}(e)^*$. Then we can write $U = U_1 \cdots U_n$ such that for $1 \leq i \leq n$ it holds that $U_i \in L_{\mathsf{SR}}(e)$. We find for $1 \leq i \leq n$ an $f_i \in \mathcal{F}$ such that $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$. By Lemma 7.28, we find an $f \in \mathcal{F}$ such that $e^* \xrightarrow{U}_{\mathsf{SR}} f$, and hence $U \in L_{\mathsf{SR}}(e^*)$. □

Lemma 7.30. *For all $e \in \mathcal{T}$, we have $L_{\mathsf{SR}}(e) = [\![e]\!]$.*

Proof. We proceed by induction on e. In the base, there are three cases to consider.

- If $e = 0$, first suppose that $U \in L_{\mathsf{SR}}(0)$. In that case, $0 \xrightarrow{U}_{\mathsf{SR}} e$ for some $e \in \mathcal{F}$. Since $0 \notin \mathcal{F}$, this means that $0 \xrightarrow{U}_{\mathsf{SR}} e$ cannot be trivial. In that case, there exists an $e' \in \mathcal{T}$ such that $0 \xrightarrow{U}_{\mathsf{SR}} e'$ is a unit run. However, this contradicts that $\delta_{\mathsf{SR}}(0, \mathtt{a}) = \emptyset$ for all $\mathtt{a} \in \Sigma$, and $\gamma_{\mathsf{SR}}(e, \phi) = \emptyset$ for all $\phi \in \mathbb{M}(\mathcal{T})$. Therefore, our assumption must be false. We conclude that $L_{\mathsf{SR}}(0) = \emptyset = [\![0]\!]$.

- If $e = 1$, suppose that $U \in L_{\mathsf{SR}}(1)$, i.e., $1 \xrightarrow{U}_{\mathsf{SR}} e$ for some $e \in \mathcal{F}$. By an argument similar to the previous case, we can argue that $1 \xrightarrow{U}_{\mathsf{SR}} e$ is trivial, and hence $U = 1$, which means that $U \in [\![1]\!]$. The other inclusion follows from the fact that $1 \in \mathcal{F}$ and $1 \xrightarrow{1}_{\mathsf{SR}} 1$.

- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, suppose $U \in L_{\mathsf{SR}}(\mathtt{a})$, i.e., $\mathtt{a} \xrightarrow{U}_{\mathsf{SR}} e$ for some $e \in \mathcal{F}$. Since $\mathtt{a} \notin \mathcal{F}$, we know $\mathtt{a} \xrightarrow{U}_{\mathsf{SR}} e$ must be non-trivial. We can then write $U = U_0 \cdot U'$, and there exists an $f \in \mathcal{T}$ such that $\mathtt{a} \xrightarrow{U_0}_{\mathsf{SR}} f$ is a unit run, and $f \xrightarrow{U'}_{\mathsf{SR}} e$. A quick glance at δ_{SR} and γ_{SR} then tells us that $f = 1$. By the previous case, we know that $f \xrightarrow{U'}_{\mathsf{SR}} e$ must be trivial; hence $U' = 1$ and $f = e$. Indeed, $\mathtt{a} \xrightarrow{U_0}_{\mathsf{SR}} f$ must be a sequential unit run, for $\gamma_{\mathsf{SR}}(\mathtt{a}, \phi) = \emptyset$ for all $\phi \in \mathbb{M}(\mathcal{T})$. Thus $U = \mathtt{b}$ and $f \in \delta_{\mathsf{SR}}(\mathtt{a}, \mathtt{b})$ for some $\mathtt{b} \in \Sigma$; by definition of δ_{SR}, it follows that $\mathtt{b} = \mathtt{a}$.

For the other inclusion, let $U = \mathtt{a}$; then $\mathtt{a} \xrightarrow{\mathtt{a}}_{\mathsf{SR}} 1$ immediately, and hence $\mathtt{a} \in L_{\mathsf{SR}}(\mathtt{a})$.

For the inductive step, all cases follow by induction and Lemma 7.29. For instance, if $e = e_1 + e_2$, then $L_{\mathsf{SR}}(e_1) = [\![e_1]\!]$ and $L_{\mathsf{SR}} e_1 = [\![e_2]\!]$, and we can derive by said lemma that

$$L_{\mathsf{SR}}(e) = L_{\mathsf{SR}}(e_1 + e_2) = L_{\mathsf{SR}}(e_1) + L_{\mathsf{SR}}(e_1) = [\![e_1]\!] + [\![e_2]\!] = [\![e]\!] \qquad \square$$

Lemma 7.31. *If $e, f \in \mathcal{T}$ such that $e \preceq_{\mathsf{SR}} f$, then $\mathsf{d}_{\|}(e) \leq \mathsf{d}_{\|}(f)$.*

Proof. It suffices to prove the claim for the rules that generate $\preceq_{\mathsf{SR}}$; this gives us three cases.

- If $e \preceq_{\mathsf{SR}} f$ because there exists an $\mathtt{a} \in \Sigma$ with $e \in \delta_{\mathsf{SR}}(f, \mathtt{a})$, then we proceed by induction on f. In the base, $f \in \Sigma$ and $e = 1$; but then $\mathsf{d}_{\|}(e) = 0 \leq \mathsf{d}_{\|}(f)$ immediately.

  For the inductive step, there are four cases to consider.

  - If $f = f_1 + f_2$, then assume without loss of generality that $e \in \delta_{\mathsf{SR}}(f_1, \mathtt{a})$. By induction, $\mathsf{d}_{\|}(e) \leq \mathsf{d}_{\|}(f_1)$; since $\mathsf{d}_{\|}(f_1) \leq \mathsf{d}_{\|}(f)$, the claim follows.
  - If $f = f_1 \cdot f_1$, then there are two subcases to consider. On the one hand, if $e \in \delta_{\mathsf{SR}}(f_1, \mathtt{a}) \fatsemi f_2$, then $e = f_1' \cdot f_2$ with $f_1' \in \delta_{\mathsf{SR}}(f_1, \mathtt{a})$. By induction, $\mathsf{d}_{\|}(f_1') \leq \mathsf{d}_{\|}(f_1)$. We then find

    $$\mathsf{d}_{\|}(e) = \max(\mathsf{d}_{\|}(f_1'), \mathsf{d}_{\|}(f_2)) \leq \max(\mathsf{d}_{\|}(f_1), \mathsf{d}_{\|}(f_2)) = \mathsf{d}_{\|}(f)$$

    On the other hand, if $e \in f_1 \star \delta_{\mathsf{SR}}(f_2, \mathtt{a})$, then $e \in \delta_{\mathsf{SR}}(f_2, \mathtt{a})$. By induction, $\mathsf{d}_{\|}(e) \leq \mathsf{d}_{\|}(f_2)$. Since $\mathsf{d}_{\|}(f_2) \leq \mathsf{d}_{\|}(f)$, the claim follows.
  - We can disregard the case where $f = f_1 \parallel f_2$, for $\delta_{\mathsf{SR}}(f, \mathtt{a}) = \emptyset$.
  - If $f = f_1^*$, then $e = f_1' \cdot f_1^*$ with $f_1' \in \delta_{\mathsf{SR}}(f_1, \mathtt{a})$. By induction, $\mathsf{d}_{\|}(f_1') \leq \mathsf{d}_{\|}(f_1)$. We then know that $\mathsf{d}_{\|}(e) = \max(\mathsf{d}_{\|}(f_1'), \mathsf{d}_{\|}(f_1)) = \mathsf{d}_{\|}(f_1) = \mathsf{d}_{\|}(f)$.

- If $e \preceq_{\mathsf{SR}} f$ because there exists a $\phi \in \mathbb{M}(\mathcal{T})$ with $e \in \gamma_{\mathsf{SR}}(f, \phi)$, we proceed by induction on f. In the base, where $f \in \{0, 1\} \cup \Sigma$, the claim holds vacuously, because $\gamma_{\mathsf{SR}}(f, \phi) = \emptyset$.

  For the inductive step, all cases except the one for parallel composition are similar to the argument above. Now, if $f = f_1 \parallel f_2$, then $e = 1$, and hence $\mathsf{d}_{\|}(e) = 0 \leq \mathsf{d}_{\|}(f)$.

- If $e \preceq_{\mathsf{SR}} f$ because $\phi \in \mathbb{M}(\mathcal{T})$ with $e \in \phi$ and $\gamma_{\mathsf{SR}}(f, \phi) \neq \emptyset$, we proceed by induction on f, showing $\mathsf{d}_{\|}(e) < \mathsf{d}_{\|}(f)$. In the base, where $f \in \{0, 1\} \cup \Sigma$, the claim holds vacuously.

  For the inductive step, there are four cases to consider.

- If $f = f_1 + f_2$, then assume without loss of generality that $\gamma_{\mathsf{SR}}(f_1, \phi) \neq \emptyset$. By induction, we have $\mathsf{d}_{\|}(e) < \mathsf{d}_{\|}(f_1)$. Since $\mathsf{d}_{\|}(f_1) \leq \mathsf{d}_{\|}(f)$, we are done.
- If $f = f_1 \cdot f_2$, then there are two subcases to consider.
  * If $\gamma_{\mathsf{SR}}(f_1, \phi) \fatsemi f_2 \neq \emptyset$, then $\gamma_{\mathsf{SR}}(f_1, \phi) \neq \emptyset$. By induction, we have that $\mathsf{d}_{\|}(e) < \mathsf{d}_{\|}(f_1)$. Since $\mathsf{d}_{\|}(f_1) \leq \mathsf{d}_{\|}(f)$, we are done.
  * If $f_1 \star \gamma_{\mathsf{SR}}(f_2, \phi) \neq \emptyset$, then $\gamma_{\mathsf{SR}}(f_2, \phi) \neq \emptyset$. By induction, we have that $\mathsf{d}_{\|}(e) < \mathsf{d}_{\|}(f_1)$. Since $\mathsf{d}_{\|}(f_1) \leq \mathsf{d}_{\|}(f)$, we are done.
- If $f = f_1 \parallel f_2$, then without loss of generality $\phi = \{\!|f_1, f_2|\!\}$ and $e = f_1$. By definition of $\mathsf{d}_{\|}(-)$, we then find that $\mathsf{d}_{\|}(e) = \mathsf{d}_{\|}(f_1) < \mathsf{d}_{\|}(f)$.
- If $f = f_1^*$, then $\gamma_{\mathsf{SR}}(f_1, \phi) \neq \emptyset$. By induction, $\mathsf{d}_{\|}(e) < \mathsf{d}_{\|}(f_1) = \mathsf{d}_{\|}(f)$. □

Lemma 7.32. *Let $e, f \in \mathcal{T}$. If f is a fork target of e in the syntactic PA, then $\mathsf{d}_{\|}(f) < \mathsf{d}_{\|}(e)$. Consequently, the syntactic PA is fork-acyclic.*

Proof. The first part of the claim was shown as the last part of the proof of Lemma 7.31. For the second part of the claim, we note that $f \preceq_{\mathsf{SR}} e$ by definition; meanwhile, $e \not\preceq_{\mathsf{SR}} e$, since if this were the case then $\mathsf{d}_{\|}(e) \leq \mathsf{d}_{\|}(f)$, which contradicts that $\mathsf{d}_{\|}(f) < \mathsf{d}_{\|}(e)$. Hence we can conclude $f \prec_{\mathsf{SR}} e$. □

Lemma 7.34. *For every $e \in \mathcal{T}$, we have that $e \in R(e)$ and $R(e)$ is support-closed. Consequently, the syntactic PA is bounded.*

Proof. It suffices to verify that $e \in R(e)$, and that for $e' \in R(e)$ the following hold:

- For all $\mathtt{a} \in \Sigma$, we have $\delta_{\mathsf{SR}}(e', \mathtt{a}) \subseteq R(e)$.
- For all $\phi \in \mathbb{M}(\mathcal{T})$, we have $\gamma_{\mathsf{SR}}(e', \phi) \subseteq R(e)$.
- If $f \in \phi \in \mathbb{M}(\mathcal{T})$ and $\gamma_{\mathsf{SR}}(e', \phi) \neq \emptyset$, then $f \in R(e)$.

We proceed by induction on e. In the base, there are two cases to consider.

- If $e \in \{0, 1\}$, then $e' = e \in R(e)$ immediately. Furthermore, note that $\delta_{\mathsf{SR}}(e, \mathtt{a}) = \emptyset$ for all $\mathtt{a} \in \Sigma$, and $\gamma_{\mathsf{SR}}(e, \phi) = \emptyset$ for all $\phi \in \mathbb{M}(\mathcal{T})$ — hence, the three conditions above hold vacuously.
- If $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, then $e \in R(e)$ as well. We consider the case where $e' = \mathtt{a}$; the case where $e' = 1$ is covered above. First, for all $\mathtt{b} \in \Sigma$, we have that $\delta_{\mathsf{SR}}(\mathtt{a}, \mathtt{b}) \subseteq \{1\}$, and hence $\delta_{\mathsf{SR}}(\mathtt{b}, \mathtt{a}) \subseteq R(e)$. Second, for all $\phi \in \mathbb{M}(\mathcal{T})$ we have that $\gamma_{\mathsf{SR}}(\mathtt{a}, \phi) = \emptyset$, and hence $\gamma_{\mathsf{SR}}(\mathtt{a}, \phi) \subseteq R(e)$. The case where $f \in \phi \in \mathbb{M}(\mathcal{T})$ with $\gamma_{\mathsf{SR}}(\mathtt{a}, \phi) \neq \emptyset$ cannot occur.

For the inductive step, there are four cases to consider.

- If $e = e_1 + e_2$, then $e \in R(e)$ by construction. To see that $R(e)$ is closed, it suffices to consider the case where $e' = e$, since $R(e_1)$ and $R(e_2)$ are closed by induction.

  - For all $\mathtt{a} \in \Sigma$, we have that $\delta_{\mathsf{SR}}(e_1 + e_2, \mathtt{a}) = \delta_{\mathsf{SR}}(e_1, \mathtt{a}) \cup \delta_{\mathsf{SR}}(e_2, \mathtt{a})$. Now, since $\delta_{\mathsf{SR}}(e_1, \mathtt{a}) \subseteq R(e_1)$ and $\delta_{\mathsf{SR}}(e_2, \mathtt{a}) \subseteq R(e_2)$ by induction, we find that $\delta_{\mathsf{SR}}(e_1 + e_2, \mathtt{a}) \subseteq R(e)$ as well.
  - For all $\phi \in \mathbb{M}(\mathcal{T})$, we have that $\gamma_{\mathsf{SR}}(e_1 + e_2, \phi) = \gamma_{\mathsf{SR}}(e_1, \phi) \cup \gamma_{\mathsf{SR}}(e_2, \phi)$. By an argument similar to the above, we find that $\gamma_{\mathsf{SR}}(e_1 + e_2, \phi) \subseteq R(e)$.
  - If $f \in \phi \in \mathbb{M}(\mathcal{T})$ such that $\gamma_{\mathsf{SR}}(e_1 + e_2, \phi) \neq \emptyset$, then either $\gamma_{\mathsf{SR}}(e_1, \phi) \neq \emptyset$ or $\gamma_{\mathsf{SR}}(e_2, \phi) \neq \emptyset$. Hence, either $\gamma_{\mathsf{SR}}(e_1, \phi) \neq \emptyset$ or $\gamma_{\mathsf{SR}}(e_2, \phi) \neq \emptyset$, and thus $f \in R(e_1) \cup R(e_2) \subseteq R(e)$.

- If $e = e_1 \cdot e_2$, then $e \in R(e)$, because $e_1 \in R(e_1)$. To see that $R(e)$ is support-closed, it suffices to consider the elements of $R(e_1) \fatsemi e_2$, since $R(e_2)$ and $R(e_1)$ are already support-closed. Let $e' = e_1' \cdot e_2$ for some $e_1' \in R(e_1)$. We consider each of the three cases.

  - For all $\mathtt{a} \in \Sigma$, we have that $\delta_{\mathsf{SR}}(e_1' \cdot e_2, \mathtt{a}) = \delta_{\mathsf{SR}}(e_1', \mathtt{a}) \fatsemi e_1 \cup e_1' \star \delta_{\mathsf{SR}}(e_2, \mathtt{a})$. Since $\delta_{\mathsf{SR}}(e_1', \mathtt{a}) \fatsemi e_1 \subseteq R(e_1)$ and $\delta_{\mathsf{SR}}(e_2, \mathtt{a}) \subseteq R(e_2)$ by induction, the claim then follows.
  - For all $\phi \in \mathbb{M}(\mathcal{T})$, we can show that $\gamma_{\mathsf{SR}}(e', \phi)$ again occurs in $R(e)$, by a similar argument.
  - If $f \in \phi \in \mathbb{M}(\mathcal{T})$ such that $\gamma_{\mathsf{SR}}(e', \phi) \neq \emptyset$, then $\gamma_{\mathsf{SR}}(e_1', \phi) \fatsemi e_2 \neq \emptyset$, or $e_1' \star \gamma_{\mathsf{SR}}(e_2, \phi) \neq \emptyset$. In the former case, $f \in R(e_1) \subseteq R(e)$, while in the latter case $f \in R(e_2) \subseteq R(e)$.

- If $e = e_1 \parallel e_2$, then $e \in R(e)$ by construction. To see that $R(e)$ is support-closed, it suffices to consider the case where $e' = e_1 \parallel e_2$.

  - For all $\mathtt{a} \in \Sigma$, we have that $\delta_{\mathsf{SR}}(e_1 \parallel e_2, \mathtt{a}) = \emptyset \subseteq R(e)$.
  - For $\phi \in \mathbb{M}(\mathcal{T})$, we have that $\gamma_{\mathsf{SR}}(e_1 \parallel e_2, \phi) \subseteq \{1\} \subseteq R(e)$ by definition.
  - For $f \in \phi \in \mathbb{M}(\mathcal{T})$ such that $\gamma_{\mathsf{SR}}(e_1, \parallel e_2, \phi) \neq \emptyset$, we have $\phi = \{\!|e_1, e_2|\!\}$, and thus $f = e_1$ or $f = e_2$. In that case, $f \in R(e_1)$ or $f \in R(e_2)$ by induction, and hence $f \in R(e)$.

- If $e = e_1^*$, then again $e \in R(e)$ by construction. To see that $R(e)$ is support-closed, it suffices to consider the case where $e' \in R(e_1) \fatsemi e_1^* \cup \{e_1^*\}$. To this end, let $e' = e_1' \cdot e_1^*$ with $e_1' \in R(e_1)$.

– For all $\mathtt{a} \in \Sigma$, we have that

$$\begin{aligned}
\delta_{\mathsf{SR}}(e_1' \cdot e_1^*, \mathtt{a}) &= \delta_{\mathsf{SR}}(e_1', \mathtt{a}) \fatsemi e_1^* \cup e_1 \star \delta_{\mathsf{SR}}(e_1^*, \mathtt{a}) \\
&\subseteq \delta_{\mathsf{SR}}(e_1', \mathtt{a}) \fatsemi e_1^* \cup \delta_{\mathsf{SR}}(e_1^*, \mathtt{a}) \\
&= \delta_{\mathsf{SR}}(e_1', \mathtt{a}) \fatsemi e_1^* \cup \delta_{\mathsf{SR}}(e_1, \mathtt{a}) \fatsemi e_1^* \\
&\subseteq R(e_1) \fatsemi e_1^* \subseteq R(e)
\end{aligned}$$

Furthermore, $\delta_{\mathsf{SR}}(e_1^*, \mathtt{a}) \subseteq R(e)$ by a similar argument.

– If $\phi \in \mathbb{M}(\mathcal{T})$, then $\gamma_{\mathsf{SR}}(e_1' \cdot e_1^*, \phi) \subseteq R(e)$ and $\gamma_{\mathsf{SR}}(e_1^*, \phi) \subseteq R(e)$ by a similar argument.

– If $f \in \phi \in \mathbb{M}(\mathcal{T})$ and $\gamma_{\mathsf{SR}}(e_1' \cdot e_1^*, \phi) \neq \emptyset$, then $\gamma_{\mathsf{SR}}(e_1', \phi) \neq \emptyset$, hence $f \subseteq R(e_1) \subseteq R(e)$. When $\gamma_{\mathsf{SR}}(e_1^*, \phi) \neq \emptyset$, we have that $f \in R(e)$ by a similar argument. □

Chapter 8

Decision Problems

Reasoning about program equivalence using algebraic laws is intuitive, but it requires some creativity on the part of the programmer. As programs grow in size, it may become quite cumbersome to mentally work out a proof using pen and paper. Moreover, we may want to integrate equivalence checking into a tool such as a compiler (to prove that an optimisation is correct), an editor (to check whether a refactoring of the code preserves semantics) or a verifier (to see whether a program matches its specification). Thus, the question arises: can we come up with an algorithm that decides whether two series-rational expressions are equivalent? To do this, we could try to write a program that searches the space of all possible proof trees for one that witnesses the desired equivalence. However, for such a program to terminate, we would have to know when to stop searching, for instance by bounding the possible size of a proof tree, and our analysis of the proof theory of series-rational expressions does not yet provide any insights in this direction.

An alternative is to not compare the series-rational expressions themselves, but their semantics; after all, completeness of $\equiv$ w.r.t. $[\![-]\!]$ tells us that checking whether $e \equiv f$ is the same as checking whether $[\![e]\!] = [\![f]\!]$. The problem here is that $[\![e]\!]$ and $[\![f]\!]$ may, in general, be infinite, and therefore will not fit in the finite memory of a machine. For this approach, we need a finite description of these languages, and finite pomset automata provide exactly such a description. As we saw in the previous section, we can in fact construct a finite and fork-acyclic pomset automaton with states that accept $[\![e]\!]$ and $[\![f]\!]$ respectively. Thus, if we could find some way to decide, provided such a PA, whether two given states accept the same language, then we would be able to decide whether the expressions represented by those states have the same semantics, and hence whether they are provably equivalent. The central topic of this chapter is a decision procedure for this problem.

The result that equivalence of series-rational expressions is decidable is not new; Laurence and Struth [LS14] proved, more generally, that equivalence of series-parallel rational expressions (i.e., including the “parallel star” operator) is decidable, based on purely syntactic reasoning. Brunet et al. [BPS17] provided a decision procedure for equivalence of sr-expressions that relies on a translation to 1-safe Petri nets. However, in addition to a (new) proof of the same result, our investigation of pomset automata in this chapter will give us new insights into pomset automata, providing new tools for the analysis of pomset automata and series-rational expressions alike.

8.1 Undecidability

Before we get started on the decision procedure, we first argue that fork-acyclicity is necessary to be able to decide whether two states in a finite PA accept the same language. To get an intuition as to why this may be the case, note that we have already demonstrated (in Example 7.10) that pomset automata can be used to accept languages (of words) that are non-rational. Indeed, pomset automata can be used to recognise a strictly more general class of languages, called *context-free languages*. To make this precise, let us first recall the notion of a context-free grammar.

Definition 8.1 (Context-free grammars)**.** A *context-free grammar* (*CFG*) is a tuple $\mathcal{G} = \langle V, \rightarrow \rangle$, where V is a finite set of symbols not appearing in Σ, called *non-terminals*, and $\rightarrow \subseteq V \times (V \cup \Sigma)^*$ is a finite relation whose elements are called *productions*. The semantics of $\mathcal{G}$, denoted $[\![-]\!]_{\mathcal{G}}$, is the least function $t : (V \cup \Sigma)^* \rightarrow 2^{\Sigma^*}$ (w.r.t. pointwise inclusion) such that the following holds:

$$\frac{}{1 \in t(1)} \qquad \frac{\mathtt{a} \in \Sigma}{\mathtt{a} \in t(\mathtt{a})} \qquad \frac{s \rightarrow w \qquad u \in t(w)}{u \in t(s)} \qquad \frac{u \in t(w) \qquad v \in t(x)}{u \cdot v \in t(w \cdot x)}$$

A language $L \subseteq \Sigma^*$ is said to be *context-free* if $L = [\![s]\!]_{\mathcal{G}}$ for some CFG $\mathcal{G} = \langle V, \rightarrow \rangle$ with $s \in V$.

Example 8.2. The language $L = \{\mathtt{a}^n \cdot \mathtt{b}^n \ : \ n \in \mathbb{N}\}$ is context-free, because it is recognised by the grammar $\mathcal{G} = \langle V, \rightarrow \rangle$, where $V = \{s\}$, and productions are given by $s \rightarrow 1$ and $s \rightarrow \mathtt{a} \cdot s \cdot \mathtt{b}$. To see that $L = [\![s]\!]_{\mathcal{G}}$, first note that if $n = 0$, then $\mathtt{a}^n \cdot \mathtt{b}^n = 1 \in [\![1]\!]_{\mathcal{G}}$; since $s \rightarrow 1$, we have that $1 \in [\![s]\!]_{\mathcal{G}}$. Next, if $n > 0$, then assume that the claim holds for $n - 1$. Since $\mathtt{a}^{n-1} \cdot \mathtt{b}^{n-1} \in [\![s]\!]_{\mathcal{G}}$, it follows that $\mathtt{a}^n \cdot \mathtt{b}^n = \mathtt{a} \cdot \mathtt{a}^n \cdot \mathtt{b}^n \cdot \mathtt{b} \in [\![\mathtt{a} \cdot s \cdot \mathtt{b}]\!]$, and since $s \rightarrow \mathtt{a} \cdot s \cdot \mathtt{b}$, it follows that $\mathtt{a}^n \cdot \mathtt{b}^n \in [\![s]\!]_{\mathcal{G}}$. The converse inclusion, i.e., that $[\![s]\!]_{\mathcal{G}} \subseteq L$ can also be proved.

To show that languages described by a CFG $\mathcal{G} = \langle V, \rightarrow \rangle$ can be accepted by a finite PA, we create an infinite but bounded PA whose states are strings over $V \cup \Sigma$; the language accepted by each of these states is intended to be the language assigned by the semantics function $[\![-]\!]_{\mathcal{G}}$.

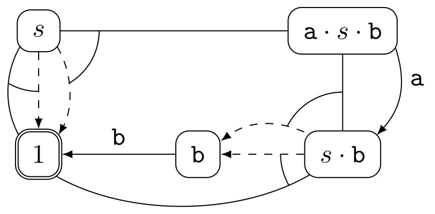


Figure 8.1: The PA support of s in the PA $A_{\mathcal{G}}$ obtained from the CFG $\mathcal{G}$ in Example 8.2.

The intuition behind the construction of this PA is that it simulates the substitution caused by the inference rules using unary forks, i.e., if we are in a state of the form $s \cdot w'$ for $s \in V$, and $s \to x$, then we can fork into x, read a word there, and then resume in w'.

Definition 8.3 (CFG to PA)**.** Let $\mathcal{G} = \langle V, \to \rangle$ be a CFG. We define the pomset automaton $A_{\mathcal{G}} = \langle Q_{\mathcal{G}}, F_{\mathcal{G}}, \delta_{\mathcal{G}}, \gamma_{\mathcal{G}} \rangle$, where $Q_{\mathcal{G}} = (V \cup \Sigma)^*$ and $F_{\mathcal{G}} = \{1\}$; also, $\delta_{\mathcal{G}}$ and $\gamma_{\mathcal{G}}$ are generated by

$$\frac{\mathtt{a} \in \Sigma \qquad w \in Q_{\mathcal{G}}}{w \in \delta_{\mathcal{G}}(\mathtt{a} \cdot w, \mathtt{a})} \qquad\qquad \frac{s \to w \qquad x \in Q_{\mathcal{G}}}{x \in \gamma_{\mathcal{G}}(s \cdot x, \{\!|w|\!\})}$$

Convention 8.4. We simplify subscripts by writing $\to_{\mathcal{G}}$ instead of $\to_{A_{\mathcal{G}}}$, et cetera.

Example 8.5. Let $\mathcal{G}$ be the CFG from Example 8.2. The support of s in $A_{\mathcal{G}}$ is depicted in Figure 8.1. Note that $A_{\mathcal{G}}$ is *not* fork-acyclic, since $s \cdot \mathtt{b} \preceq \mathtt{a} \cdot s \cdot \mathtt{b}$, while $\mathtt{a} \cdot s \cdot \mathtt{b}$ is a fork target of $s \cdot \mathtt{b}$.

To see that $[\![s]\!]_{\mathcal{G}} \subseteq L_{\mathcal{G}}(s)$, first note that since $\mathtt{1} \xrightarrow{1}_{\mathcal{G}} 1$ and $\mathtt{b} \in \gamma_{\mathcal{G}}(s \cdot \mathtt{b}, \{\!|1|\!\})$ we have $s \cdot \mathtt{b} \xrightarrow{1}_{\mathcal{G}} \mathtt{b}$. Furthermore, since $\mathtt{b} \xrightarrow{\mathtt{b}}_{\mathcal{G}} 1$, also $s \cdot \mathtt{b} \xrightarrow{\mathtt{b}}_{\mathcal{G}} 1$. Now, because $\mathtt{a} \cdot s \cdot \mathtt{b} \xrightarrow{\mathtt{a}}_{\mathcal{G}} s \cdot \mathtt{b}$, we have that $\mathtt{a} \cdot s \cdot \mathtt{b} \xrightarrow{\mathtt{a} \cdot \mathtt{b}}_{\mathcal{G}} 1$. More generally, one can show that for $n \geq 1$ it holds that $\mathtt{a} \cdot s \cdot \mathtt{b} \xrightarrow{\mathtt{a}^n \cdot \mathtt{b}^n}_{\mathcal{G}} 1$. Since $1 \in \gamma_{\mathcal{G}}(s, \{\!|\mathtt{a} \cdot s \cdot \mathtt{b}|\!\})$, we then find for $n \geq 1$ that $s \xrightarrow{\mathtt{a}^n \cdot \mathtt{b}^n}_{\mathcal{G}} 1$. Lastly, since $1 \in \gamma_{\mathcal{G}}(s, \{\!|1|\!\})$, we also have $s \xrightarrow{1}_{\mathcal{G}} 1$, and hence $s \xrightarrow{\mathtt{a}^n \cdot \mathtt{b}^n}_{\mathcal{G}} 1$ for all $n \in \mathbb{N}$. The converse inclusion, i.e., that $L_{\mathcal{G}}(s) \subseteq [\![s]\!]_{\mathcal{G}}$, can also be shown.

The PA we considered in the example above had a fork cycle. In general, one can prove that any CFG with a cyclic dependency between non-terminals (e.g., s appears on the right-hand side of a production from s, as is the case for $\mathcal{G}$ discussed above) yields a PA with a fork cycle. For our purposes, however, it suffices to observe that there exists CFGs $\mathcal{G}$ for which $A_{\mathcal{G}}$ is fork-acyclic.

In the same example, we also saw that the support of the state s in $A_{\mathcal{G}}$ was finite. This is true in general; the key insight to prove this is that if $w, x \in Q_{\mathcal{G}}$ such that $w \preceq_{\mathcal{G}} x$, then either the length of w is bounded by the length of x (because w can reach x by a series of unit transitions) or by the length of some right-hand side to a production rule in $\mathcal{G}$ (because x can be reached from a fork target, which is always such a right-hand side). Formally, we can then prove the following.

Lemma 8.6. *Let $\mathcal{G} = \langle V, \to \rangle$ be a CFG. Then $A_\mathcal{G}$ is bounded.*

In the example we considered, the PA $A_\mathcal{G}$ faithfully implemented the language of a given non-terminal in the grammar. We can show that this is the case in general, by arguing as follows.

Lemma 8.7. *Let $\mathcal{G} = \langle V, \to \rangle$ be a CFG. Then for $w \in (V \cup \Sigma)^*$ we have that $L_\mathcal{G}(w) = [\![w]\!]_\mathcal{G}$.*

Proof. For the inclusion from left to right, it suffices to show that $L_\mathcal{G}$ satisfies the conditions that define $[\![-]\!]_\mathcal{G}$. There are four rules to consider.

- First, we observe that since $1 \in F_\mathcal{G}$, we have that $1 \in L_\mathcal{G}(1)$ immediately.
- Second, we note that $1 \in \delta_\mathcal{G}(\mathtt{a}, \mathtt{a})$, and thus $\mathtt{a} \xrightarrow{\mathtt{a}}_\mathcal{G} 1$, which means that $\mathtt{a} \in L_\mathcal{G}(\mathtt{a})$.
- Next, suppose that $s \to w$ and that $u \in L_\mathcal{G}(w)$. In that case, we know that $w \xrightarrow{u}_\mathcal{G} 1$. Since $1 \in \gamma_\mathcal{G}(s, \{\!|w|\!\})$, it then follows that $s \xrightarrow{u}_\mathcal{G} 1$, and hence $u \in L_\mathcal{G}(s)$.
- Lastly, suppose that $u \in L_\mathcal{G}(w)$ and $v \in L_\mathcal{G}(x)$. In that case, $w \xrightarrow{u}_\mathcal{G} 1$ and $v \xrightarrow{x}_\mathcal{G} 1$. A simple argument by induction on the construction of $\to_\mathcal{G}$ then shows that $w \cdot x \xrightarrow{u}_\mathcal{G} v$, and hence $w \cdot x \xrightarrow{u \cdot v}_\mathcal{G} 1$. This allows us to conclude that $u \cdot v \in L_\mathcal{G}(w \cdot x)$.

For the other inclusion, we note that by Lemma 7.36 it suffices to show that for $w \in Q_\mathcal{G}$, we have:

$$\frac{w \in F_\mathcal{G}}{1 \in [\![w]\!]_\mathcal{G}} \qquad \frac{\mathtt{a} \in \Sigma \qquad w' \in \delta_\mathcal{G}(w, \mathtt{a})}{\mathtt{a} \cdot [\![w']\!]_\mathcal{G} \subseteq [\![w]\!]_\mathcal{G}} \qquad \frac{w' \in \gamma_\mathcal{G}(w, \{\!|y_1, \dots, y_n|\!\})}{([\![y_1]\!]_\mathcal{G} \parallel \cdots \parallel [\![y_n]\!]_\mathcal{G}) \cdot [\![w']\!]_\mathcal{G} \subseteq [\![w]\!]_\mathcal{G}}$$

We validate each of the rules separately:

- Suppose that $w \in F_\mathcal{G}$. In that case, $w = 1$ by definition of $F_\mathcal{G}$; since $1 \in [\![1]\!]_\mathcal{G}$, we are done.
- Suppose $x \in [\![w']\!]_\mathcal{G}$ with $w' \in \delta_\mathcal{G}(w, \mathtt{a})$. In that case we have $w = \mathtt{a} \cdot w'$, and hence $\mathtt{a} \cdot x \in [\![w]\!]_\mathcal{G}$.
- Suppose $x \in [\![w']\!]_\mathcal{G}$ such that $w' \in \gamma_\mathcal{G}(w, \{\!|y_1, \dots, y_m|\!\})$, and that for $1 \leq i \leq m$ we have $x_i \in [\![y_i]\!]_\mathcal{G}$. By construction of $\gamma_\mathcal{G}$, we then have that $m = 1$ and we obtain $s \to y_1$ such that $w = s \cdot w'$. Since $x_1 \in [\![s]\!]_\mathcal{G}$ and $x \in [\![w']\!]_\mathcal{G}$, it follows that $x_1 \cdot x \in [\![s \cdot w']\!]_\mathcal{G} = [\![w]\!]_\mathcal{G}$. □

Remark 8.8. Conversely, one can argue that if $A = \langle Q, F, \delta, \gamma \rangle$ is a finite pomset automaton where all forks are *unary* — i.e., such that whenever $\gamma(q, \phi) \neq \emptyset$, also $|\phi| = 1$ — then the languages accepted by A can be described by a CFG. In [KBL+19], CFGs are augmented with parallel composition to describe pomset languages proper. It can be shown that any CFG in this format can be implemented by a PA, and that every PA can be represented by such a CFG. For simplicity, we restrict our discussion here to the well-known context-free grammars over words.

Having shown that a finite pomset automaton can implement a context-free language, we are ready to prove that equivalence of states in finite pomset automata (possibly with a fork cycle) is undecidable. To this end, we first recall the following well-known result about CFGs.

Theorem 8.9 [BPS61]**.** *The following problem is undecidable:*

CFG-EQUIV: *Given a CFG $\mathcal{G} = \langle V, \rightarrow \rangle$ with $v, v' \in V$. Does $[\![v]\!]_\mathcal{G} = [\![v']\!]_\mathcal{G}$ hold?*

Corollary 8.10. *The following problem is undecidable:*

PA-EQUIV: *Given a finite PA $A = \langle Q, F, \delta, \gamma \rangle$ with $q, q' \in Q$. Does $L_A(q) = L_A(q')$ hold?*

Proof. Suppose PA-EQUIV were decidable. Then we could decide CFG-EQUIV by computing $A_\mathcal{G}$ and restricting it to the support of v and v', and checking whether the languages accepted by v and v' in the resulting finite pomset automaton are the same. By Lemmas 7.20 and 8.7, this holds precisely when $[\![v]\!]_\mathcal{G} = [\![v']\!]_\mathcal{G}$, and hence our decision procedure would be correct. Since CFG-EQUIV is undecidable by Theorem 8.9, we conclude that PA-EQUIV cannot be decidable, either. □

Remark 8.11. In [BPS61], it is also shown that, given a CFG $\mathcal{G} = \langle V, \rightarrow \rangle$ with $v \in V$, it is undecidable whether $[\![v]\!]_\mathcal{G}$ is a rational language. Since all series-rational languages of words are in fact rational languages, the problem of deciding whether a given state in a given PA accepts a series-rational language is therefore also undecidable. This also explains why fork-acyclicity is a sufficient but not necessary condition for the translation of PAs to sr-expressions: any precondition that completely characterises correctness of such a construction would be undecidable.

8.2 Well-structuredness

As we saw in the previous section, language equivalence of states in a pomset automaton is undecidable. In this section, we therefore focus on fork-acyclic pomset automata, and ask whether language equivalence is decidable there. To fully appreciate the complexity of this problem, we start by illustrating the intricacies of PAs through a series of examples. These show how pomset automata with heterogeneous structures can still accept the same language. Any procedure to decide language equivalence for fork-acyclic PAs must take such cases into account.

Example 8.12 (Run confusion)**.** The pomset automaton depicted in Figure 8.1 has the state $s \cdot \mathtt{b}$, which may accept the pomset $\mathtt{b}$ by forking into the state 1, accepting and resuming computation in the state $\mathtt{b}$, where the pomset $\mathtt{b}$ can be read to arrive in the accepting state 1. Thus, $s \cdot \mathtt{b}$ accepts $\mathtt{b}$ by means of a composite run, whereas $\mathtt{b}$ accepts the same pomset by means of a sequential unit run.

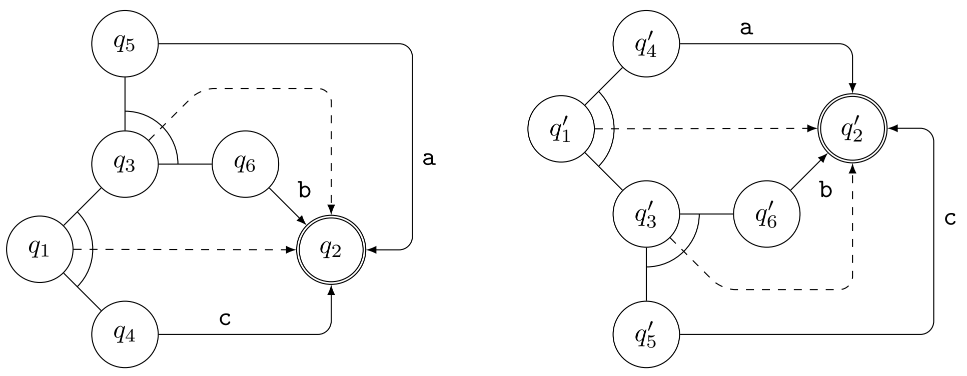


Figure 8.2: Associativity of parallelism in pomset automata.

More generally, a state could accept a more complicated pomset by means of a composite run, while another state accepts the same pomset with a parallel unit run.

Alternatively, we may also consider forks into the empty multiset ⧄; in a sense, these are analogous to ϵ-transitions in NFAs, since the parallel composition of zero multisets is 1. This could, for instance, allow a non-accepting state to accept the empty pomset.

From this example, we can take away that different types of runs may be labelled by the same pomset, especially when fork targets are allowed to be accepting states, or when forking into the empty or singleton multiset is permitted. The next example is about how nested forks may spread out behaviour of a state in structurally different but semantically equivalent ways.

Example 8.13 (Associativity)**.** Consider the PA in Figure 8.2, where both q_1 and q'_1 accept the language $\{\mathtt{a} \parallel \mathtt{b} \parallel \mathtt{c}\}$. In the transition from q_1 to q_2, the pomset $\mathtt{a} \parallel \mathtt{b}$ is contributed by q_3, and $\mathtt{c}$ comes from q_4, while in the transition from q'_1 to q'_2, we obtain $\mathtt{a}$ from q'_4, and $\mathtt{b} \parallel \mathtt{c}$ from q'_3. The language accepted by q_3 is distinct from, and in fact incomparable with, the languages accepted by q'_3 and q'_4. Hence, to compare $L_A(q_1)$ to $L_A(q'_1)$, we must consider not only the fork targets, but also the fork targets of those fork targets, for second-level forks that point to an accepting state.

We can counteract the patterns exhibited in Examples 8.12 and 8.13, by preventing the structural patterns that enable this kind of behaviour. This leads to the following definition.

Definition 8.14 (Well-structured)**.** A pomset automaton $A = \langle Q, F, \delta, \gamma \rangle$ is *well-structured* if for all $q, q' \in Q$ and all $\phi \in \mathbb{M}(Q)$ with $q' \in \phi$ and $\gamma(q, \phi) \neq \emptyset$, the following three conditions hold:

$$|\phi| \geq 2 \qquad\qquad q' \notin F \qquad\qquad \forall \phi' \in \mathbb{M}(Q).\ \gamma(q', \phi') \cap F = \emptyset$$

Example 8.15. The PAs in Figures 7.1 and 7.3a are well-structured. On the other hand, a pomset automaton with a state q such that $\gamma(q, \boxslash) \neq \emptyset$ — or $\gamma(q, \{\!|r|\!\}) \neq \emptyset$ for some state r — is not well-structured (by the first condition). The PA in Figure 7.2b is not well-structured (by the second condition) because $\gamma(q_1, \{\!|q_3, q_4|\!\}) \neq \emptyset$, while $q_4 \in F$. Finally, the PA in Figure 8.2 is not well-structured (by the third condition) because $\gamma(q_1, \{\!|q_3, q_4|\!\}) \neq \emptyset$, while $q_2 \in \gamma(q_3, \{\!|q_5, q_6|\!\}) \cap F$.

Well-structuredness is a relatively mild restriction; it does not prevent us from expressing the behaviour of fork-acyclic PAs that we have seen up to this point. For instance, the behaviour of the PA discussed in Example 8.13 (see Figure 8.2) can be expressed using a ternary fork, and the behaviour of the PA discussed in Example 8.12 (see Figure 7.2b) can be obtained by a sequential transition. Indeed, as we shall formally prove in Section 8.3, any fork-acyclic pomset automaton can be implemented by a well-nested and fork-acyclic pomset automaton. For now, we focus on deciding equivalence of states in well-structured and fork-acyclic finite pomset automata. In the remainder of this section, we fix one such pomset automaton $A = \langle Q, F, \delta, \gamma \rangle$.

The benefit of well-structured PAs is that run type is uniquely determined by the pomset that labels it. Furthermore, because forks cannot be nested and fork targets cannot accept, the pomsets U accepted by a fork target must all be parallel primes (i.e., if $U = V \parallel W$, then $V = 1$ or $W = 1$).

Lemma 8.16. *If A is well-structured and $q \xrightarrow{U}_A q'$, then the following hold:*

(i) $q \xrightarrow{U}_A q'$ is trivial if and only if U is empty.

(ii) $q \xrightarrow{U}_A q'$ is a sequential unit run if and only if U is primitive.

(iii) $q \xrightarrow{U}_A q'$ is a composite run if and only if U is sequential.

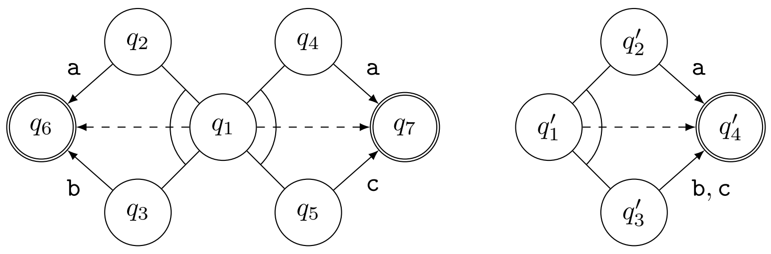
(iv) $q \xrightarrow{U}_A q'$ is a parallel unit run if and only if U is parallel.

If q is a fork target and $q' \in F$, then U is a parallel prime.

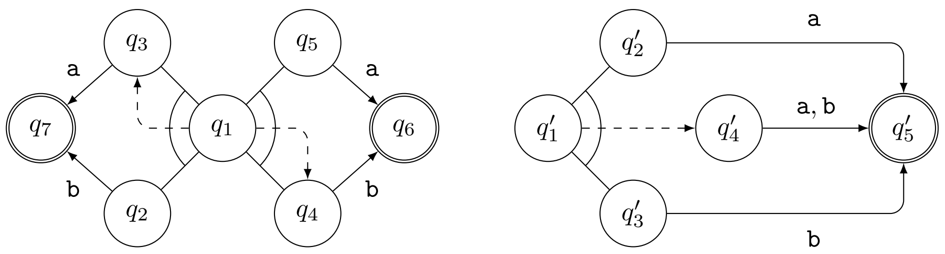
Even for well-structured PAs, there are still structural factors that can confound language equivalence of states, and with which a decision procedure for equivalence will need to reckon.

Example 8.17 (Distributivity I)**.** In Figure 8.3a, we have the state q_1 which may fork into q_2 and q_3, as well as q_4 and q_5. Now, q_1 accepts the language $\{\mathtt{a} \parallel \mathtt{b}, \mathtt{a} \parallel \mathtt{c}\}$, where the former behaviour stems from forking into q_2 and q_3, but the latter is obtained by forking into q_4 and q_5. On the other hand, we have the state q_1', which may fork into q_2' and q_3'; here, q_1' accepts the same language as q_1, but the two pomsets are due to the *same* fork. In a sense, this is a consequence of the distributivity of parallel composition over union of pomset languages:

$$L_A(q_1) = \mathtt{a} \parallel \mathtt{b} + \mathtt{a} \parallel \mathtt{c} = \mathtt{a} \parallel (\mathtt{b} + \mathtt{c}) = L_A(q_1')$$



(a) Distributivity of parallelism over union.



(b) Distributivity of sequential composition over union.

Figure 8.3: PAs where q_1 and q_1' accept the same language, with different transition structures.

This example illustrates how behaviour of language-equivalent states may be spread out across different parallel transitions, and that this division may differ locally. The last example stems from an implicit kind of non-determinism that is supported by PAs (that we touched upon in Remark 7.5), that can be a result of overlap between the languages accepted by fork targets.

Example 8.18 (Distributivity II)**.** In Figure 8.3b, q_1 we can read $\mathtt{a} \parallel \mathtt{b}$ to arrive in q_3 or q_4. From that point on, q_3 can read $\mathtt{a}$ to reach an accepting state, while q_4 can read $\mathtt{b}$ to do the same. In contrast, q_1' can read $\mathtt{a} \parallel \mathtt{b}$ to arrive in q_4' only, whence it can read $\mathtt{a}$ or $\mathtt{b}$ to arrive in q_5' and accept. Nevertheless, q_1 and q_1' accept the same language. Even though the behaviour implemented by the forks from q_1 is the same, where they land is not uniquely determined. In a sense, this is a consequence of distributivity of sequential composition over union of pomset languages:

$$L_A(q_1) = (\mathtt{a} \parallel \mathtt{b}) \cdot \mathtt{a} + (\mathtt{a} \parallel \mathtt{b}) \cdot \mathtt{b} = (\mathtt{a} \parallel \mathtt{b}) \cdot (\mathtt{a} + \mathtt{b}) = L_A(q_1')$$

Examples 8.17 and 8.18 illustrate that there are many structurally different ways to express parallel behaviour in a pomset automaton. It is not so clear how to choose one particular way of representing parallel behaviour. Instead, we design our algorithm to directly deal with the these phenomena. To understand how this can be done, we first shift perspective from deciding language equivalence to finding out which states do and do not overlap in terms of their language [LS14].

Definition 8.19 (Atoms)**.** Let A be a PA or an NFA, with states Q and $\alpha \subseteq Q$. We write

$$L_A(\alpha) = \left(\bigcap \{L_A(q) \ : \ q \in \alpha\}\right) \setminus \left(\bigcup \{L_A(q) \ : \ q \in Q \setminus \alpha\}\right)$$

When $L_A(\alpha) \neq \emptyset$, we say that α is an *atom* of A; we write At_A for the set of atoms of A.

Example 8.20. In the PA in Figure 8.3a, $\{q_3, q_3'\}$ is an atom, as is $\{q_5, q_3'\}$; the set $\{q_2, q_4, q_2'\}$ is also an atom. In fact, these are all atoms that contain fork targets of Figure 8.3a.

The PA in Figure 8.3b has $\{q_3, q_5, q_2', q_4'\}$ as an atom; similarly, $\{q_2, q_4, q_3', q_4'\}$ is an atom. These are again all of the atoms that contain fork targets in Figure 8.3b.

The following lemma shows how atoms can be used to decide language equivalence.

Lemma 8.21. *Let A be a PA or NFA with states q_1 and q_2. Then $L_A(q_1) = L_A(q_2)$ if and only if for all $\alpha \in \mathsf{At}_A$ it holds that $q_1 \in \alpha$ precisely when $q_2 \in \alpha$.*

Thus, if we can compute the set of atoms a PA then we can decide language equivalence of states. As it turns out, this is possible if the PA is finite, fork-acyclic and well-structured.

Lemma 8.22. *If A is finite, fork-acyclic, and well-structured, then At_A is computable.*

Proof. We proceed by induction on D_A. In the base, where $D_A = 0$, we know that $Q = \emptyset$, since if $q \in Q$, then $D_A(q) \geq 1$; it is not hard to see that $\emptyset$ is the only atom in this case.[1]

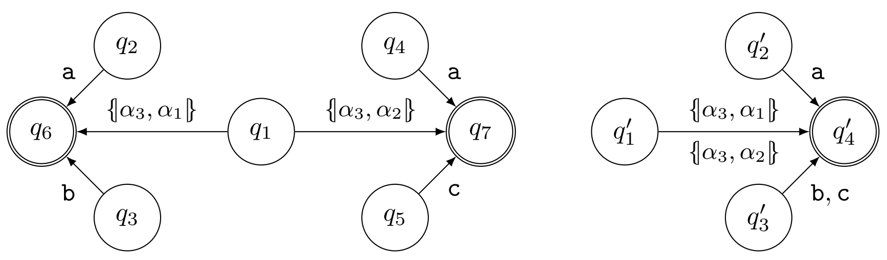
For the inductive step, let $D_A > 0$ and suppose that the claim holds for PAs of strictly lower depth. We choose $Q' = \{q \in Q \ : \ D_A(q) < D_A\}$, and note that Q' is support-closed: if $q' \preceq_A q \in Q'$, then $D_A(q') \leq D_A(q)$, and hence $q' \in Q'$. By Lemma 7.20, we can restrict A to obtain $A[Q']$, which, by construction, is of strictly lower depth than A. By induction, we can compute $\mathsf{At}_{A[Q']}$.

To compute At_A, we shall construct an NFA $A' = \langle Q, \delta', F\rangle$ whose atoms are precisely those of A; the claim then follows because we can compute the atoms of an NFA using a standard algorithm such as the one by Brzozowski and Tamm [BT14]. The intuition behind A' is to copy the δ-transitions of A, and encode the γ-transitions by transitions labelled with symbols built from $\mathsf{At}_{A[Q']}$. More precisely, we will have transitions labelled by symbols from the following alphabet:

$$\Delta = \Sigma \cup \left\{ \{\!\{\alpha_1, \dots, \alpha_n\}\!\} \in \mathbb{M}(\mathsf{At}_{A[Q']}) \ : \ \begin{array}{c} \exists q \in Q, \ q_1 \in \alpha_1, \ \dots, \ q_n \in \alpha_n. \\ \gamma(q, \{\!\{q_1, \dots, q_n\}\!\}) \neq \emptyset \end{array} \right\}$$

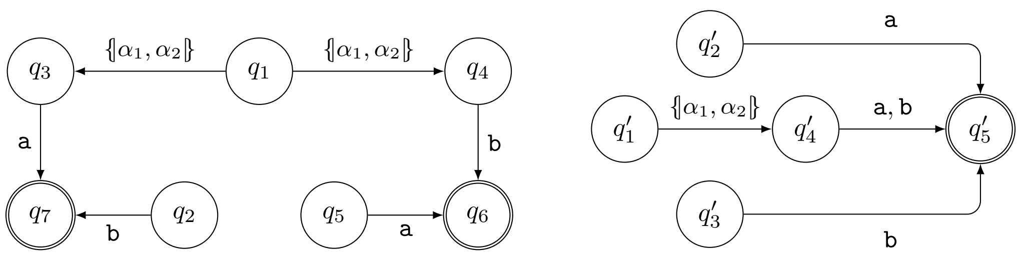
Here, we assume without loss of generality that the two sets are disjoint, i.e., that none of the multisets are already symbols in Σ. This alphabet is finite because $\mathsf{At}_{A[Q']}$ is finite, and because by definition of pomset automata there are finitely many multisets ϕ s.t. $\gamma(q, \phi) \neq \emptyset$.

[1]The empty intersection is assumed to be the set of all sp-pomsets SP.



q_2
q_4
q'_2
a
a
a
$\{\!|\alpha_3, \alpha_1|\!\}$
$\{\!|\alpha_3, \alpha_2|\!\}$
$\{\!|\alpha_3, \alpha_1|\!\}$
q_6
q_1
q_7
q'_1
q'_4
$\{\!|\alpha_3, \alpha_2|\!\}$
b
c
b, c
q_3
q_5
q'_3

(a) NFA created in inductive step of atom computation for Figure 8.3a.



a
q'_2
$\{\!|\alpha_1, \alpha_2|\!\}$
$\{\!|\alpha_1, \alpha_2|\!\}$
q_3
q_1
q_4
$\{\!|\alpha_1, \alpha_2|\!\}$
a, b
a
b
q'_1
q'_4
q'_5
q_7
q_2
q_5
q_6
b
a
q'_3
b

(b) NFA created in inductive step of atom computation for Figure 8.3b.

Figure 8.4: Examples of PAs obtained in the inductive step of atom computation.

We define the transition function of our NFA, $\delta' : Q \times \Delta \to 2^Q$, as follows.

$$\delta'(q, \mathtt{a}) = \delta(q, \mathtt{a}) \qquad \delta'(q, \{\!|\alpha_1, \dots, \alpha_n|\!\}) = \bigcup \{\gamma(q, \{\!|q_1, \dots, q_n|\!\}) \ : \ q_1 \in \alpha_1, \ \cdots, \ q_n \in \alpha_n\}$$

Example 8.23. Let A be the PA in Figure 8.3a. In Example 8.20, we saw that $\alpha_1 = \{q_3, q'_3\}$, $\alpha_2 = \{q_5, q'_3\}$ and $\alpha_3 = \{q_2, q_4, q'_2\}$ are the atoms that contain fork targets. The resulting NFA is drawn in Figure 8.4a. There, we see that $q_6 \in \delta'(q_2, \mathtt{a})$ because $q_6 \in \delta(q_2, \mathtt{a})$. Furthermore, $q_6 \in \delta'(q_1, \{\!|\alpha_3, \alpha_1|\!\})$ because $q_6 \in \gamma(q_1, \{\!|q_2, q_3|\!\})$ and $q_2 \in \alpha_3$ while $q_3 \in \alpha_1$.

Example 8.24. Alternatively, let A be the PA in Figure 8.3b. In Example 8.20, we found atoms $\alpha_1 = \{q_3, q_5, q'_2, q'_4\}$ and $\alpha_2 = \{q_2, q_4, q'_3, q'_4\}$. The resulting NFA is drawn in Figure 8.4b. There, we see that $q_7 \in \delta'(q_1, \{\!|\alpha_1, \alpha_2|\!\})$ because $q_7 \in \gamma(q_1, \{\!|q_3, q_2|\!\})$ with $q_3 \in \alpha_1$ and $q_2 \in \alpha_2$.

It remains to show that the atoms of A are the same as those of A'. To this end, we relate the languages accepted by A' to those accepted A, using the substitution $\zeta : \Delta \to 2^{\mathsf{SP}(\Sigma)}$, given by:

$$\zeta(\mathtt{a}) = \{\mathtt{a}\} \qquad \zeta(\{\!|\alpha_1, \dots, \alpha_n|\!\}) = L_A(\alpha_1) \parallel \cdots \parallel L_A(\alpha_n)$$

We need the following two technical properties of ζ. The first of these relates the languages accepted by the states of the NFA A' to the languages accepted by the states of the PA A by means of ζ.

Fact 8.25. *For all $q \in Q$, it holds that $L_A(q) = \zeta(L_{A'}(q))$.*

The second property that we need says that ζ is compatible with a number of operators on language; this is a consequence of the fact that A is well-structured, and hence if $\{\!|\alpha_1, \dots, \alpha_n|\!\} \in \Delta$, then $n \geq 2$ and $1 \notin L_A(\alpha_i)$ for all $1 \leq i \leq n$. Since each state in α_i is a fork target, and the pomsets accepted by fork targets must be parallel primes, $\zeta(\{\!|\alpha_1, \dots, \alpha_n|\!\})$ must consist of sequential primes.

Fact 8.26. *The following hold for all $L, K \subseteq \mathsf{SP}(\Delta)$:*

$$\zeta(L \cup K) = \zeta(L) \cup \zeta(K) \qquad \zeta(L \cap K) = \zeta(L) \cap \zeta(K)$$
$$\zeta(L) \setminus \zeta(K) = \zeta(L \setminus K) \qquad \zeta(L) = \emptyset \iff L = \emptyset$$

Let $\alpha \subseteq Q$; we can then use the facts above to calculate that

$$\begin{aligned}
\zeta(L_{A'}(\alpha)) &= \zeta\Big(\Big(\bigcap \{L_{A'}(q) \;:\; q \in \alpha\}\Big) \setminus \Big(\bigcup \{L_{A'}(q) \;:\; q \in Q \setminus \alpha\}\Big)\Big) && (\text{def. } L_{A'} \text{ on } 2^Q) \\
&= \Big(\bigcap \{\zeta(L_{A'}(q)) \;:\; q \in \alpha\}\Big) \setminus \Big(\bigcup \{\zeta(L_{A'}(q)) \;:\; q \in Q \setminus \alpha\}\Big) && (\text{Fact 8.26}) \\
&= \Big(\bigcap \{L_A(q) \;:\; q \in \alpha\}\Big) \setminus \Big(\bigcup \{L_A(q) \;:\; q \in Q \setminus \alpha\}\Big) && (\text{Fact 8.25}) \\
&= L_A(\alpha) && (\text{def. } L_A \text{ on } 2^Q)
\end{aligned}$$

To wrap up, suppose α is an atom of A, i.e., $L_A(\alpha) \neq \emptyset$. By the derivation above, this holds precisely when $\zeta(L_{A'}(\alpha)) \neq \emptyset$; by Fact 8.26, this is equivalent to having that $L_{A'}(\alpha) \neq \emptyset$, i.e., α is an atom of A'. This shows that $\mathsf{At}_A = \mathsf{At}_{A'}$; since we can compute the latter, the claim follows. □

Lemmas 8.21 and 8.22 now give us a preliminary decidability result for equivalence in PAs.

Theorem 8.27. *The problem* PA-EQUIV *is decidable for well-structured and fork-acyclic PAs.*

Remark 8.28. When deciding equivalence of two states in a well-structured and fork-acyclic PA $A = \langle Q, F, \delta, \gamma \rangle$, we do not need to compute the atoms of A proper. Instead, we can compute the atoms of the PA one level below (i.e., $A[Q']$ where $Q' = \{q \in Q \;:\; D_A(q) < D_A\}$) and check equivalence of q and q' in the NFA A' built on Q using these atoms (as in the proof of Lemma 8.22).

8.3 Normalisation

In the previous section, we saw that language equivalence for states in a well-structured, fork-acyclic and finite pomset automaton is decidable. In this section, we discharge the precondition that the pomset automaton is well-structured, by showing that we can construct a well-structured, fork-acyclic and finite PA that implements a given fork-acyclic and finite PA.

We first decompose the definition of well-structured automata into three simpler properties.

Definition 8.29 (n-forking)**.** Let $n \in \mathbb{N}$. A pomset automaton A is *n-forking* if for every state $q \in Q$ and every multiset of states $\phi \in \mathbb{M}(Q)$ such that $\gamma(q, \phi) \neq \emptyset$ we have $|\phi| \geq n$.

Definition 8.30 (Parsimony)**.** A pomset automaton A is said to be *parsimonious* if, whenever $p \in Q$ and $q \in \phi \in \mathbb{M}(Q)$ such that $\gamma(p, \phi) \neq \emptyset$, we have that $1 \notin L_A(q)$.

Definition 8.31 (Flat-branching)**.** A pomset automaton A is *flat-branching* if for all states $p, q \in Q$ and every multiset $\phi \in \mathbb{M}(Q)$, if $\gamma(q, \phi) \neq \emptyset$ and $p \in \phi$ then

$$\forall \psi \in \mathbb{M}(Q).\ \gamma(p, \psi) \cap F = \emptyset.$$

Example 8.32. The PAs displayed in Figures 7.1, 7.2b, 7.3, 7.4, 8.2 and 8.3 are 2-forking, but the ones in Figures 7.2a and 8.1 are not. The automata in Figure 8.3 are parsimonious, but the ones in Figures 7.2b and 7.3 are not; for instance, in Figure 7.2b, we have that $\gamma(q_1, \{\!|q_3, q_4|\!\}) \neq \emptyset$, while $1 \in L_A(q_4)$. The automata in Figures 7.3 and 8.3 are flat-branching, but the one in Figure 8.2 is not; in particular, there we have that $\gamma(q_1, \{\!|q_3, q_4|\!\}) \neq \emptyset$, while $q_2 \in \gamma(q_3, \{\!|q_5, q_6|\!\}) \cap F$.

One can prove that the above properties indeed guarantee well-structuredness.

Lemma 8.33. *A PA A is well-structured if it is 2-forking, parsimonious, and flat-branching.*

Thus, objective can be fulfilled by converting a given fork-acyclic and finite PA into an equivalent fork-acyclic and finite PA that is also 2-forking, parsimonious and flat-branching. Note that these properties can be at cross purposes: for instance, ensuring parsimony may introduce forks of smaller sizes, which could make it so that the PA is no longer 2-forking. Similarly, eliminating forks into the empty multiset may introduce new accepting states, which can invalidate flat-branching.

It should be clear, then, that ensuring all of these properties hold at the same time requires some care. The remainder of this section is devoted to a series of transformations, each of which establishes one property while maintaining the ones guaranteed by the previous step. More precisely, our construction to ensure well-structuredness will consist of the following four steps:

(1) we first show how to implement A with a parsimonious pomset automaton A_0;

(2) then we discuss how to implement A with a 1-forking pomset automaton A_1;

(3) we proceed to show how to implement A with a 2-forking pomset automaton A_2.

(4) finally, we show that A can be implemented by a flat-branching pomset automaton A_3;

Since each of these transformations preserves the established properties (e.g., A_2 is still 1-forking and parsimonious), we end up with a pomset automaton that implements A and is 2-forking, parsimonious, and flat-branching, and hence well-structured by Lemma 8.33. Furthermore, since A is finite (and therefore bounded), it follows that the resulting PA must also be bounded, and can therefore be restricted to a finite pomset automaton by Lemma 7.20.

Before we get into the weeds, we discuss some technical properties that will help simplify the constructions. First, note that we shall need to rule out a form of silent transitions that can occur by forking into states accepting the empty pomset (if the pomset automaton is not parsimonious) or by forking into the empty multiset (if the PA is not 1-forking). The result is a non-trivial run labelled by the empty pomset. To reason about such transitions, we observe the following:

Lemma 8.34. *For any finite pomset automaton A, the predicate $p \xrightarrow{1}_A q$ is decidable.*

Another useful notion for this section will be that of *weak implementation*. Essentially, a weak implementation of a PA is another PA where the behaviour of each state of the first PA is spread out across a set of states, rather than just one (as is the case for implementation).

Definition 8.35. A pomset automaton A' *weakly implements* a PA A if the following hold:

(i) for every state q in A there is a finite set of states Q_q in A' s.t. $L_A(q) = \bigcup_{x \in Q_q} L_{A'}(x)$, and

(ii) if A is fork-acyclic (respectively bounded), then so is A'.

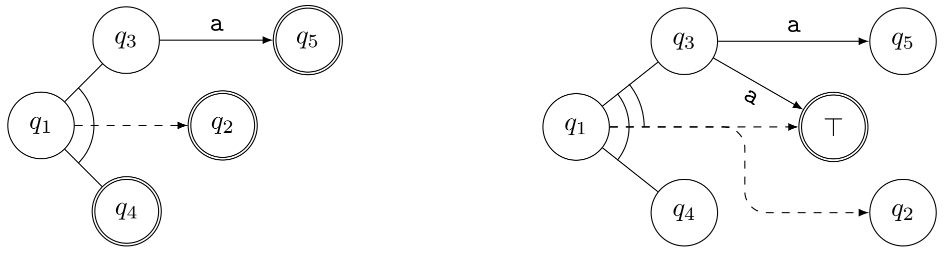
As it turns out, to prove that there exists a pomset automaton implementing A that satisfies some of the properties above, it suffices to find one that weakly implements A with the same property. We shall make use of this in the first three constructions to follow.

Lemma 8.36. *If a PA A' weakly implements a PA A, then there exists a PA A'' implementing A. Furthermore, if A' is n-forking (respectively flat-branching, parsimonious), then so is A''.*

8.3.1 Parsimony

Let $A = \langle Q, F, \delta, \gamma \rangle$ be a finite, fork-acyclic PA; we want to implement A with a fork-acyclic and parsimonious PA A_0. There are two key ideas to this translation:

- We introduce a new state $\top$, which will be the sole accepting state of the PA; in fact, it will be the only state accepting the empty pomset. This state will not have any outgoing transitions, so its language is exactly $\{1\}$. We will modify the transition functions, such that if a transition in A can lead to a state that accepts 1, it can also lead to $\top$ in A_0.



(a) A PA A that is not parsimonious. (b) Part of the PA A_0 obtained from A.

Figure 8.5: Example of construction to ensure parsimony.

- To ensure preservation of successful runs, we need to add parallel transitions to mitigate the previous modification. Thus, if $q \in Q$ can fork into $\phi \sqcup \psi \in \mathbb{M}(Q)$ to reach $q' \in Q$, i.e., $q' \in \gamma(q, \phi \sqcup \psi)$, and all states in ψ accept the empty pomset in A, then we make sure that, in A_0, q can fork into ϕ to reach q', simulating the acceptance of 1 from states in ψ.

Doing so, we obtain a pomset automaton weakly implementing A — as a matter of fact, if $1 \notin L_A(q)$, then $L_A(q) = L_{A_0}(q)$, and otherwise $L_A(q) = L_{A_0}(q) \cup L_{A_0}(\top)$. Since $\top$ cannot be a fork target, and any other state cannot accept the empty pomset, this new pomset automaton is parsimonious. Finiteness and fork-acyclicity are also maintained by this construction.

Definition 8.37 (A_0)**.** The pomset automaton A_0 is given by the tuple $\langle Q_0, F_0, \delta_0, \gamma_0 \rangle$ where $Q_0 = Q \cup \{\top\}$ (with $\top \notin Q$), and $F_0 = \{\top\}$. Furthermore, δ_0 is generated by the rules

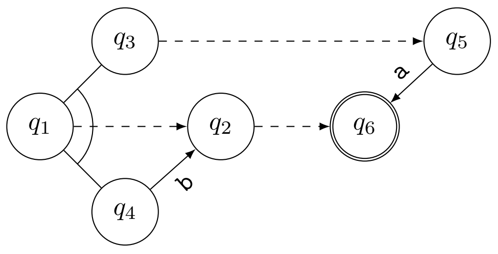
$$\frac{q' \in \delta(q, \mathtt{a})}{q' \in \delta_0(q, \mathtt{a})} \qquad\qquad \frac{q' \in \delta(q, \mathtt{a}) \qquad 1 \in L_A(q')}{\top \in \delta_0(q, \mathtt{a})}$$

Also, γ_0 is generated by the following rules for all $q \in Q$ and $\phi \in \mathbb{M}(Q)$:

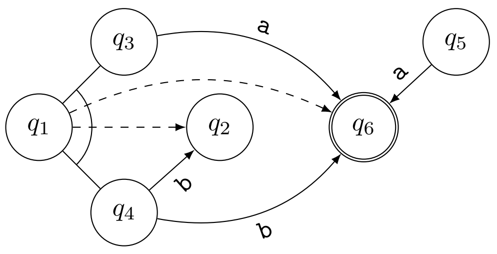
$$\frac{\begin{array}{c} q' \in \gamma(q, \phi \sqcup \psi) \\ \forall r \in \psi.\ 1 \in L_A(r) \end{array}}{q' \in \gamma_0(q, \phi)} \qquad\qquad \frac{\begin{array}{cc} q' \in \gamma(q, \phi \sqcup \psi) & 1 \in L_A(q') \\ \phi \neq \boxslash & \forall r \in \psi.\ 1 \in L_A(r) \end{array}}{\top \in \gamma_0(q, \phi)}$$

Lastly, $\delta_0(\top, \mathtt{a}) = \emptyset$ for all $\mathtt{a} \in \Sigma$ and $\gamma_0(\top, \phi) = \emptyset$ for all $\phi \in \mathbb{M}(Q_0)$.

Example 8.38. Consider the pomset automaton A depicted in Figure 8.5a, which is not parsimonious. The resulting pomset automaton A_0 is drawn in Figure 8.5b. Since $q_5 \in \delta(q_3, \mathtt{a})$ in A, we have $q_5 \in \delta_0(q_3, \mathtt{a})$ by the first rule generating δ_0; since $1 \in L_A(q_5)$, we also have $\top \in \delta_0(q_3, \mathtt{a})$ by the second rule generating δ_0. Furthermore, since $1 \in L_A(q_4)$ and $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$ while $1 \in L_A(q_2)$, we have that $\top \in \gamma_0(q_1, \{\!|q_3, q_4|\!\})$ as well as $\top \in \gamma_0(q_1, \{\!|q_3|\!\})$ by the second rule generating γ_0.



(a) A PA A with nullary forks.



(b) The PA A_1 obtained from A.

Figure 8.6: Example of nullary fork removal.

Note that even though in the example above A was 2-forking, A_0 was not, as a result of the fact that $\top \in \gamma_0(q_1, \{\!|q_3|\!\})$. We will remedy the appearance of unary forks later on.

Our construction is correct, in the following sense.

Lemma 8.39. *A_0 is parsimonious and weakly implements A.*

8.3.2 Removing nullary forks

Let $A = \langle Q, F, \delta, \gamma \rangle$ be a finite, parsimonious, and fork-acyclic pomset automaton. We want to (weakly) implement A with a 1-forking PA A_1 while maintaining finiteness, parsimony and fork-acyclicity. As mentioned, the *nullary* forks that we want to eliminate — i.e., those where $q' \in \gamma(q, \boxslash)$ — essentially furnish silent transitions $q \xrightarrow{1}_A q'$, analogous to classic NFAs. We proceed eliminate these by saturating the transition functions with $\xrightarrow{1}_A$ [HU79, Chapter 2.4].

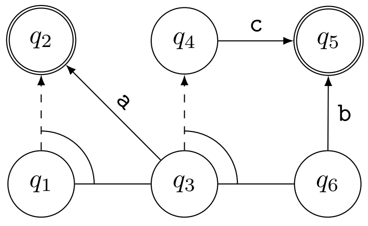
Definition 8.40 (A_1)**.** The pomset automaton A_1 is defined to be $\langle Q, F, \delta_1, \gamma_1 \rangle$, where δ_1 and γ_1 are generated by the following inference rules for all $\mathtt{a} \in \Sigma$ and $\phi \in \mathbb{M}(Q)$ with $\phi \neq \boxslash$.

$$\frac{p \xrightarrow{1}_A q \qquad r \in \delta(q, \mathtt{a}) \qquad r \xrightarrow{1}_A s}{s \in \delta_1(p, \mathtt{a})} \qquad\qquad \frac{p \xrightarrow{1}_A q \qquad r \in \gamma(q, \phi) \qquad r \xrightarrow{1}_A s}{s \in \gamma_1(p, \phi)}$$

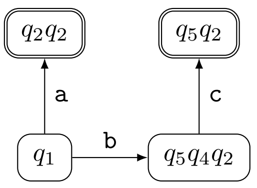
Example 8.41. Suppose A is the pomset automaton in Figure 8.6a. This pomset automaton has two nullary forks: $q_5 \in \gamma(q_3, \boxslash)$ and $q_6 \in \gamma(q_2, \boxslash)$. We have drawn the pomset automaton A_1 obtained from A in Figure 8.6b. Here, $q_6 \in \delta_1(q_3, \mathtt{a})$, because $q_3 \xrightarrow{1}_A q_5$, $q_6 \in \delta(q_5, \mathtt{a})$ and $q_6 \xrightarrow{1}_A q_6$. Similarly, $q_6 \in \gamma_1(q_1, \{\!|q_3, q_4|\!\})$ because $q_1 \xrightarrow{1}_A q_1$, $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\})$ and $q_2 \xrightarrow{1}_A q_6$.

We conclude by stating correctness of our translation, in the following sense.

Lemma 8.42. *A_1 is 1-forking, parsimonious, and weakly implements A.*



(a) A PA A with unary forks.



(b) Part of the automaton A_2 obtained from A.

Figure 8.7: Example of unary fork removal.

8.3.3 Removing unary forks

We now show that, given a fork-acyclic, finite, parsimonious and 1-forking PA $A = \langle Q, F, \delta, \gamma \rangle$, we can implement it using a 2-forking PA A_2 that retains the properties of A. The main idea is to simulate unary forks by keeping a "call stack" in the state. When A follows a unary fork, e.g., $q' \in \gamma(q, \{\!|r|\!\})$, A_2 will push q' on the stack to "remember" where we should continue after completing the computation in r; once we reach an accepting state, the transitions of q' will become available. Because we put no upper limit on the size of the stack, the resulting PA will have infinitely many states; this is not a problem, however, since we prove that A_2 weakly implements A, and hence A_2 is bounded when A is — meaning that A_2 can be made finite if necessary.

To keep track of this call stack, we need to know which unary forks can occur in sequence from any given state. This is captured by the following.

Definition 8.43. The relation $\uparrow$, is the least subset of $Q \times Q^*$ satisfying the following rules:

$$\frac{}{q \uparrow q} \qquad\qquad \frac{r \uparrow w \qquad q' \in \gamma(q, \{\!|r|\!\})}{q \uparrow w \cdot q'}$$

Intuitively, if $q \uparrow q_1 \cdots q_n$, then q can perform a series of unary forks leading to state q_1; once the computation starting in q_1 reaches an accepting state q_1', we can pop q_1' off the stack and continue in q_2, and so on. The first rule covers the case where no fork takes place, while the second rule allows to extend an existing series of forks with one more.

Example 8.44. Suppose A is the PA in Figure 8.7a. We first note that $q_6 \uparrow q_6$ by the first rule; hence, since $q_4 \in \gamma(q_3, \{\!|q_6|\!\})$, we have $q_3 \uparrow q_6 \cdot q_4$ by applying the second rule. Furthermore, since $q_2 \in \gamma(q_1, \{\!|q_3|\!\})$, we find that $q_1 \uparrow q_6 \cdot q_4 \cdot q_2$, again by the second rule. Hence, q_1 can fork into q_6, and after completing a computation there and in q_4, we can carry on in q_2.

We can now define our transformation, as follows.

Definition 8.45 (A_2)**.** The PA A_2 is defined to be $\langle Q_2, F_2, \delta_2, \gamma_2 \rangle$, where $Q_2 = Q^*$ and $F_2 = F^*$. Also, δ_2 and γ_2 are generated by the following rules for $\mathtt{a} \in \Sigma$ and $\phi \in \mathbb{M}(Q)$ with $|\phi| \geq 2$:

$$\frac{q \uparrow r \cdot w \qquad q' \in \delta(r, \mathtt{a})}{q' \cdot w \cdot x \in \delta_2(q \cdot x, \mathtt{a})} \qquad\qquad \frac{w' \in \delta_2(w, \mathtt{a}) \qquad q \in F}{w' \in \delta_2(q \cdot w, \mathtt{a})}$$

$$\frac{q \uparrow r \cdot w \qquad q' \in \gamma(r, \phi)}{q' \cdot w \cdot x \in \gamma_2(q \cdot x, \phi)} \qquad\qquad \frac{w' \in \gamma_2(w, \phi) \qquad q \in F}{w' \in \gamma_2(q \cdot w, \phi)}$$

The first rule allows us to look at the state q on top of the stack, and see where it can fork to; if the state r where we end up in allows a δ-transition to q', we push q' onto the stack, along with the unresolved states w gained from the unary fork.[2] The second rule says that we can also look at states further up the stack, provided that they are preceded by accepting states only. This allows us to pop accepting states off the stack, while continuing in the next state. The latter two rules work analogously to the first two rules, but for parallel unit transitions.

Example 8.46. Let A be the pomset automaton in Figure 8.7a. We have drawn the support of q_1 in the pomset automaton A_2 obtained from A in Figure 8.7b. There, we have $q_2 \cdot q_2, q_5 \cdot q_2 \in F_2$ because $q_2, q_5 \in F$. Also, since $q_1 \uparrow q_3 \cdot q_2$ (see Example 8.44) and $q_2 \in \delta(q_3, \mathtt{a})$, we have $q_2 \cdot q_2 \in \delta_2(q_1, \mathtt{a})$ by the first rule above. Furthermore, since $q_1 \uparrow q_6 \cdot q_4 \cdot q_2$ (see Example 8.44) and $q_5 \in \delta(q_6, \mathtt{b})$, we have $q_5 \cdot q_4 \cdot q_2 \in \delta_2(q_1, \mathtt{b})$, again by the first rule above. Lastly, $q_4 \uparrow q_4$, so $q_5 \cdot q_2 \in \delta_2(q_4 \cdot q_2, \mathtt{c})$ by the first rule. Since $q_5 \in F$, we find that $q_5 \cdot q_2 \in \delta_2(q_5 \cdot q_4 \cdot q_2, \mathtt{c})$ by the second rule.

Our transformation is correct, in the following sense.

Lemma 8.47. *A_2 is parsimonious, 2-forking, and implements A.*

8.3.4 Flat-branching

In this section, we enforce flat-branching. We start from a PA A that is assumed to be fork-acyclic, finite, parsimonious, and 2-forking, and we want to construct a flat-branching PA A_3 that weakly implements A, while keeping the properties of A.

The first idea of this construction is fairly obvious: to remove chains of forks while retaining the same language, we will saturate the parallel transitions by unfolding every possible chain of forks. For instance, if $q \in \gamma(p, \{\!|q_1, q_2|\!\})$ and $\gamma(q_1, \{\!|r_1, r_2|\!\}) \cap F \neq \emptyset$, we want to have $q \in \gamma_3(p, \{\!|r_1, r_2, q_2|\!\})$. Fork-acyclicity is instrumental for this construction to terminate, as fork cycles could introduce infinitely many $\phi \in \mathbb{M}(Q)$ such that $\gamma(p, \phi) \neq \emptyset$. We make this idea formal as follows:

[2]Note that since $q \uparrow q$, we have that $q' \cdot x \in \delta_2(q \cdot x, \mathtt{a})$ whenever $q' \in \delta(q, \mathtt{a})$.

Definition 8.48. We define $\blacktriangleleft$ as the smallest reflexive relation on $\mathbb{M}(Q)$ satisfying:

$$\frac{\gamma(p,\chi) \cap F \neq \emptyset \qquad \phi \blacktriangleleft \psi \sqcup \{\!|p|\!\}}{\phi \blacktriangleleft \psi \sqcup \chi}$$

Intuitively, $\phi \blacktriangleleft \psi$ when a fork into ϕ can be expanded to a fork into ψ, by forking from one or more of its states, provided the new fork reaches an accepting state.

Example 8.49. Recall the left half of the pomset automaton in Figure 8.2, depicted in Figure 8.8a. Since $\blacktriangleleft$ is reflexive, we have $\{\!|q_3|\!\} \blacktriangleleft \{\!|q_3|\!\}$. Then, because $\gamma(q_3, \{\!|q_5, q_6|\!\}) \cap F \neq \emptyset$, we have that $\{\!|q_3|\!\} \blacktriangleleft \{\!|q_5, q_6|\!\}$ applying the rule. Similarly, we have that $\{\!|q_1|\!\} \blacktriangleleft \{\!|q_3, q_4|\!\}$. Combining these two using the rule above then tells us that $\{\!|q_1|\!\} \blacktriangleleft \{\!|q_5, q_6, q_4|\!\}$. Thus, any fork into $\{\!|q_1|\!\}$ to reach some q' can be expanded to a fork into $\{\!|q_5, q_6, q_4|\!\}$ to reach q'.

Next, we want to make sure that the original forks cannot be executed in succession, by forcing all forks to expand maximally before continuing with some non-forking transition. The main idea is to split each state q into $q^{\mathbf{s}}$ and $q^{\mathbf{P}}$. The state $q^{\mathbf{s}}$ will ensure that $\gamma_3(q^{\mathbf{s}}, \phi) \cap F = \emptyset$ for any multiset ϕ, i.e., no forks are allowed. We leverage this property to get flat-branching, by making sure that for any state $p \in Q_3$ of the new pomset automaton, $\gamma_3(p, \phi) \neq \emptyset$ implies that every state in ϕ is of the $q^{\mathbf{s}}$ variety. On the other hand, from the state $q^{\mathbf{P}}$, one cannot perform δ-transitions, and also for any multiset ϕ we have $\gamma_3(q^{\mathbf{P}}, \phi) \subseteq \{\top\}$, where $\top$ is the unique accepting state of A_3.

Definition 8.50 (A_3)**.** The pomset automaton A_3 is the quadruple $\langle Q_3, F_3, \delta_3, \gamma_3 \rangle$, where

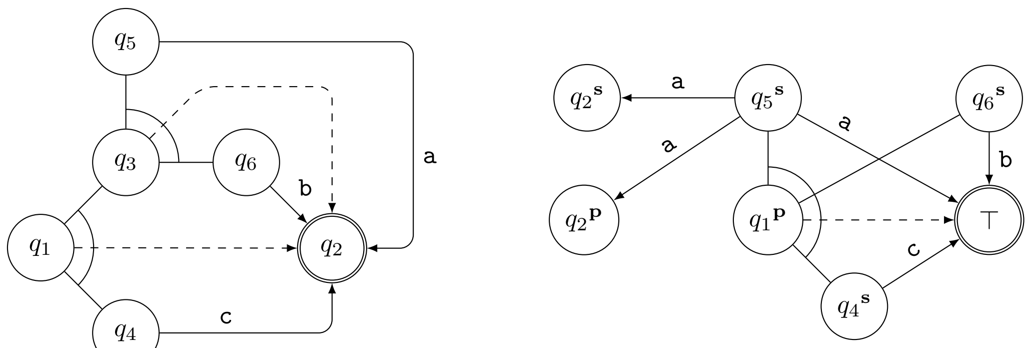
$$Q_3 = \{q^{\mathbf{P}} \;:\; q \in Q\} \cup \{q^{\mathbf{s}} \;:\; q \in Q\} \cup \{\top\} \qquad\qquad F_3 = \{\top\}$$

with $\top$ a fresh state, such that for all $\mathtt{a} \in \Sigma$ and $\phi \in \mathbb{M}(Q_3)$ we have $\delta_3(\top, \mathtt{a}) = \gamma_3(\top, \phi) = \emptyset$. Furthermore, the action of δ_3 and γ_3 on states different from $\top$ is generated by the following rules for all $\mathtt{a} \in \Sigma$ and all $\psi, \phi \in \mathbb{M}(Q)$ with $\psi \blacktriangleleft \phi$:

$$\frac{q \in \delta(p, \mathtt{a})}{q^{\mathbf{s}}, q^{\mathbf{P}} \in \delta_3(p^{\mathbf{s}}, \mathtt{a})} \qquad \frac{\delta(p, \mathtt{a}) \cap F \neq \emptyset}{\top \in \delta_3(p^{\mathbf{s}}, \mathtt{a})} \qquad \frac{q \in \gamma(p, \psi)}{q^{\mathbf{s}}, q^{\mathbf{P}} \in \gamma_3(p^{\mathbf{s}}, \phi^{\mathbf{s}})} \qquad \frac{\gamma(p, \psi) \cap F \neq \emptyset}{\top \in \gamma_3(p^{\mathbf{P}}, \phi^{\mathbf{s}})}$$

in which $\phi^{\mathbf{s}} = \{\!|{q_1}^{\mathbf{s}}, \dots, {q_n}^{\mathbf{s}}|\!\}$ whenever $\phi = \{\!|q_1, \dots, q_n|\!\}$.

Example 8.51. Let A be the PA in Figure 8.8a. Part of the support of ${q_1}^{\mathbf{P}}$ is drawn in Figure 8.8b. There, we find that, since $q_2 \in \gamma(q_1, \{\!|q_3, q_4|\!\}) \cap F$ and $\{\!|q_3, q_4|\!\} \blacktriangleleft \{\!|q_4, q_5, q_6|\!\}$ (refer to the previous example), also $\top \in \gamma_3({q_1}^{\mathbf{P}}, \{\!|{q_4}^{\mathbf{s}}, {q_5}^{\mathbf{s}}, {q_6}^{\mathbf{s}}|\!\})$ by the last rule. Furthermore, since $q_2 \in \delta(q_5, \mathtt{a}) \cap F$, we have that $\top \in \delta_3({q_5}^{\mathbf{s}}, \mathtt{a})$ by the second rule, but also ${q_2}^{\mathbf{s}}, {q_2}^{\mathbf{P}} \in \delta_3({q_5}^{\mathbf{s}}, \mathtt{a})$, by the first rule.



(a) A PA A with nested forks. (b) Part of the PA A_3 obtained from A.

Figure 8.8: Example of construction to ensure flat-branching.

Not drawn are the transitions $q_2{}^{\mathbf{s}}, q_2{}^{\mathbf{p}} \in \gamma_3(q_1{}^{\mathbf{s}}, \{\!| q_3{}^{\mathbf{s}}, q_4{}^{\mathbf{s}} |\!\})$ which would result from applying the third rule. These do not contribute to the language: they lead to states without accepting runs.

We conclude this transformation by stating the desired correctness.

Lemma 8.52. *A_3 is 2-forking, parsimonious, flat-branching, and weakly implements A.*

8.3.5 Wrapping up

Lemmas 8.39, 8.42, 8.47 and 8.52 combine with Lemma 8.36, to prove this section's objective:

Theorem 8.53. *Let A be a finite and fork-acyclic PA. We can construct a finite and fork-acyclic PA A' that is well-structured and implements A.*

As a corollary, we can strengthen Theorem 8.27 to the following:

Corollary 8.54. *The problem* PA-EQUIV *is decidable for fork-acyclic PAs.*

Because of the Kleene theorem derived in the previous chapter, we also have a new proof of Theorem 3.42, which we record as the following corollary:

Corollary 8.55. *Given $e, f \in \mathcal{T}$, it is decidable whether $[\![e]\!] = [\![f]\!]$.*

Summary of this chapter Because PAs can recognise context-free languages, language equivalence of states is undecidable in general. On the other hand, for fork-acyclic PAs satisfying certain structural properties we could derive an algorithm to decide language equivalence. We furthermore proved that these structural properties can be guaranteed while preserving both fork-acyclicity and the accepted language, thus establishing that language equivalence is decidable for fork-acyclic PAs.

8.A Proof that PA for a CFG is bounded

Lemma 8.6. *Let $\mathcal{G} = \langle V, \to \rangle$ be a CFG. Then $A_\mathcal{G}$ is bounded.*

Proof. For $w \in Q_\mathcal{G}$, let $R(w)$ be the smallest set satisfying the following rules

$$\frac{}{w \in R(w)} \qquad \frac{s \to w}{x \in R(w)} \qquad \frac{x \cdot y \in R(w)}{y \in R(w)}$$

In other words, $R(w)$ is the smallest suffix-closed subset of $Q_\mathcal{G}$ that contains w as well as the right-hand side of any production in $\mathcal{G}$. It should be clear that this set is finite. It now suffices to show that $R(w)$ is support-closed — after all, if this is the case, then $R(w)$ must contain $\pi_\mathcal{G}(w)$, and hence the latter is finite. It suffices to check the following for all $x \in R(w)$:

- Suppose that $\mathtt{a} \in \Sigma$; we should show that $\delta_\mathcal{G}(x, \mathtt{a}) \subseteq R$. By construction of $\delta_\mathcal{G}$, we know that if $x' \in \delta_\mathcal{G}(x, \mathtt{a})$, then $x = \mathtt{a} \cdot x'$. Since R is suffix-closed, it follows that $x \in R$.
- Suppose that $\phi \in \mathbb{M}(Q_\mathcal{G})$; we should show that $\gamma_\mathcal{G}(w, \phi) \subseteq R$. The argument here is similar: if $x' \in \gamma_\mathcal{G}(x, \phi)$, then x is a suffix of w, and hence $x \in R$.
- Suppose that x is a fork target of w; we should show that $x \in R$. By construction of $\gamma_\mathcal{G}$, we have that x is the right-hand side of some production in $\mathcal{G}$, and thus x must occur in R. $\square$

8.B Proofs towards correctness of decision procedure

Lemma 8.16. *If A is well-structured and $q \xrightarrow{U}_A q'$, then the following hold:*

(i) $q \xrightarrow{U}_A q'$ is trivial if and only if U is empty.

(ii) $q \xrightarrow{U}_A q'$ is a sequential unit run if and only if U is primitive.

(iii) $q \xrightarrow{U}_A q'$ is a composite run if and only if U is sequential.

(iv) $q \xrightarrow{U}_A q'$ is a parallel unit run if and only if U is parallel.

If q is a fork target and $q' \in F$, then U is a parallel prime.

Proof. We start by treating (i) in detail. Here, we note that the implication from left to right holds by definition. For the implication from right to left, we proceed by induction on the construction of $q \xrightarrow{U}_A q'$. In the base, $q \xrightarrow{U}_A q'$ is already trivial, in which case $U = 1$ holds immediately. For the inductive step, there are two cases to consider.

- Suppose that $U = V \cdot W$ and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$, then $V = W = 1$. By induction, we then know that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$ are trivial, and hence $q = q'' = q'$. We conclude that $q \xrightarrow{U}_A q'$ must also be trivial.
- Suppose that $U = U_1 \parallel \cdots \parallel U_n$ and there exist $r_1, \ldots, r_n \in Q$ as well as $r'_1, \ldots, r'_n \in F$ such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r'_i$, and furthermore $q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\})$. Then necessarily $U_1, \ldots, U_n = 1$. Since A is well-structured, we also know that $r_1, \ldots, r_n \notin F$, and furthermore that $n \geq 2$. However, by induction, we know that for $1 \leq i \leq n$ it holds that $r_i \xrightarrow{U_i}_A r'_i$ is trivial, and hence it would follow that $r_i = r'_i$. We have now reached a contradiction, for $r_1 \in F$ while also $r_1 \notin F$. We can therefore disregard this case.

We treat the implications from left to right for the remaining claims as follows.

(ii) If $q \xrightarrow{U}_A q'$ is a sequential unit run, then U is primitive by definition of sequential unit runs.

(iii) Suppose that $q \xrightarrow{U}_A q'$ is composite. We then know that $U = V \cdot W$ and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$ are nontrivial. By (i), we then know that V and W must be non-empty, and hence U is sequential.

(iv) Suppose that $q \xrightarrow{U}_A q'$ is a parallel unit run. Then $U = U_1 \parallel \cdots \parallel U_n$ and there exist $r_1, \ldots, r_n \in Q$ and $r'_1, \ldots, r'_n \in F$, such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r'_i$, and furthermore $q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\})$. By the premise that A is well-structured, we know that $r_1, \ldots, r_n \notin F$ and $n \geq 2$. Thus, for $1 \leq i \leq n$, the run $r_i \xrightarrow{U_i}_A r'_i$ is non-trivial, and hence U_i is non-empty. From this and the fact that $n \geq 2$, we can conclude that U is parallel.

The implications in the other direction follow from the fact that a pomset cannot be both sequential and parallel, or empty and primitive, and so forth. For instance, if U is primitive, then $q \xrightarrow{U}_A q'$ can be a sequential unit run only, for if it were trivial then U would be empty, if it were composite then U would be sequential, and if it were a parallel unit run then U would be parallel. □

Lemma 8.21. *Let A be a PA or NFA with states q_1 and q_2. Then $L_A(q_1) = L_A(q_2)$ if and only if for all $\alpha \in \mathsf{At}_A$ it holds that $q_1 \in \alpha$ precisely when $q_2 \in \alpha$.*

Proof. First, suppose $L_A(q_1) = L_A(q_2)$, and let $\alpha \in \mathsf{At}$ with $q_1 \in \alpha$. Then $q_2 \in \alpha$, because otherwise $L_A(\alpha) \subseteq L_A(q_1) \setminus L_A(q_2) = \emptyset$, even though α is an atom. Similarly, $q_2 \in \alpha$ implies $q_1 \in \alpha$.

For the other implication, let $U \in L_A(q_1)$. Choose $\alpha = \{q \in Q : U \in L_A(q)\}$, and note that α is an atom, since $U \in L_A(\alpha)$. Because $q_1 \in \alpha$, we find $q_2 \in \alpha$, and thus $U \in L_A(q_2)$. This shows that $L_A(q_1) \subseteq L_A(q_2)$; the other inclusion follows by symmetry. □

Fact 8.25. *For all $q \in Q$, it holds that $L_A(q) = \zeta(L_{A'}(q))$.*

Proof. It suffices to prove that, for $q, q' \in Q$ as well as $U \in \mathsf{SP}(\Sigma)$, it holds that $q \xrightarrow{U}_A q'$ is a unit run if and only if there exists an $\mathtt{a} \in \Delta$ with $U \in \zeta(\mathtt{a})$ and $q' \in \delta'(q, \mathtt{a})$; straightforward inductive arguments on run length (for the forward inclusion, using Lemma 7.8) and word length (for the backwards inclusion) complete the proof. First, suppose $q \xrightarrow{U}_A q'$ is a unit run. We have two cases.

- If $q \xrightarrow{U}_A q'$ is a sequential unit run, then $U = \mathtt{b}$ for some $\mathtt{b} \in \Sigma$, and $q' \in \delta(q, \mathtt{b})$. We can then choose $\mathtt{a} = \mathtt{b}$ to find that $q' \in \delta'(q, \mathtt{a})$ by definition of δ'.
- If $q \xrightarrow{U}_A q'$ is a parallel unit run, then $U = U_1 \parallel \cdots \parallel U_n$ and we obtain $q_1, \dots, q_n \in Q$ such that $q' \in \gamma(q, \{\!|q_1, \dots, q_n|\!\})$ as well as $U_i \in L_A(q_i)$. For $1 \leq i \leq n$, we then choose $\alpha_i = \{r \in Q' : U_i \in L_A(r)\} \in \mathsf{At}_{A[Q']}$, and set $\mathtt{a} = \{\!|\alpha_1, \dots, \alpha_n|\!\} \in \Delta$. We now find that $U \in \zeta(\mathtt{a})$ as well as $q' \in \delta'(q, \{\!|\alpha_1, \dots, \alpha_n|\!\})$.

Conversely, suppose $\mathtt{a} \in \Delta$ such that $U \in \zeta(\mathtt{a})$ and $q' \in \delta'(q, \mathtt{a})$. We have two cases to consider.

- If $\mathtt{a} \in \Sigma$, then $q' \in \delta(q, \mathtt{a})$ and $U = \mathtt{a}$. Hence $q \xrightarrow{U}_A q'$ is a sequential unit run.
- If $\mathtt{a} = \{\!|\alpha_1, \dots, \alpha_n|\!\}$, then $q' \in \gamma(q, \{\!|q_1, \dots, q_n|\!\})$ with $q_i \in \alpha_i$. Since $U \in \zeta(\mathtt{a})$, also $U = U_1 \parallel \cdots \parallel U_n$ with $U_i \in L_A(q_i)$. Thus, $q \xrightarrow{U}_A q'$ is a parallel unit run. □

Fact 8.26. *The following hold for all $L, K \subseteq \mathsf{SP}(\Delta)$:*

$$\zeta(L \cup K) = \zeta(L) \cup \zeta(K) \qquad \zeta(L \cap K) = \zeta(L) \cap \zeta(K)$$
$$\zeta(L) \setminus \zeta(K) = \zeta(L \setminus K) \qquad \zeta(L) = \emptyset \iff L = \emptyset$$

Proof. The first equality holds for substitutions in general. For the remainder of the claim, we start by observing that ζ has the following properties:

(a) for all $\mathtt{a} \in \Delta$, we have that $\zeta(\mathtt{a})$ consists of sequential primes exclusively, and

(b) for all $\mathtt{a}, \mathtt{b} \in \Delta$, we have that $\zeta(\mathtt{a}) \cap \zeta(\mathtt{b}) \neq \emptyset$ if and only if $\mathtt{a} = \mathtt{b}$, and

(c) for all $U \in \mathsf{SP}(\Sigma)$, there exists at most one $w \in \Delta^*$ such that $U \in \zeta(w)$.

We prove these claims as follows:

(a) If $\mathtt{a} \in \Delta$ and $U \in \zeta(\mathtt{a})$, then $U = \mathtt{b}$ for some $\mathtt{b} \in \Sigma$, or $\mathtt{a} = \{\!|\alpha_1, \dots, \alpha_n|\!\}$ and $U \in L_A(\alpha_1) \parallel \cdots \parallel L_A(\alpha_n)$ for atoms $\alpha_1, \dots, \alpha_n \in \mathsf{At}_{A[Q']}$. In the former case, U is primitive, which means it is a sequential prime. In the latter case, we know that none of these atoms contain the empty set and $n \geq 2$ (since A is well-structured); it follows that U is a sequential prime.

(b) Furthermore, suppose $\mathtt{a}, \mathtt{b} \in \Delta$ such that $U \in \zeta(\mathtt{a})$ and $U \in \zeta(\mathtt{b})$. By (a), we know that U is either primitive and $\mathtt{a}, \mathtt{b} \in \Sigma$, or U is parallel and $\mathtt{a}, \mathtt{b} \in \Delta \setminus \Sigma$.

In the former case, it follows that $\mathtt{a} = U = \mathtt{b}$ by definition of ζ. In the latter case, we know that $U = U_1 \parallel \cdots \parallel U_n$ and $U = U'_1 \parallel \cdots \parallel U'_{n'}$ as well as $\mathtt{a} = \{\!|\alpha_1, \ldots, \alpha_n|\!\}$ and $\mathtt{b} = \{\!|\alpha'_1, \ldots, \alpha'_{n'}|\!\}$, such that for $1 \leq i \leq n$ we have that $U_i \in L_A(\alpha_i)$ and for $1 \leq i \leq n'$ we have that $U'_i \in L_A(\alpha'_i)$. Because A is well-structured, each of the U_i and U'_i is a parallel prime; by Lemma 3.17, we then find that $n = n'$, and without loss of generality we have that $U_i = U'_i$. Since $U_i \in L_A(\alpha_i) \cap L_A(\alpha'_i)$, it follows that $\alpha_i = \alpha'_i$, and hence $\mathtt{a} = \mathtt{b}$.

(c) Suppose that $w, w' \in \Sigma^*$ such that $U \in \zeta(w)$ and $U \in \zeta(w')$. This means that we can write $w = \mathtt{a}_1 \cdots \mathtt{a}_n$ and $w' = \mathtt{a}'_1 \cdots \mathtt{a}'_{n'}$, as well as $U = U_1 \cdots U_n$ and $U = U'_1 \cdots U'_{n'}$ such that for $1 \leq i \leq n$ we have $U_i \in \zeta(\mathtt{a}_i)$, and for $1 \leq j \leq n'$ we have $U'_j \in \zeta(\mathtt{a}'_j)$. Indeed, since these U_i and U'_j are sequential prime (by the first property of ζ), Lemma 3.17, tells us that $n = n'$, and that for $1 \leq i \leq n$ we have that $U_i = U'_i$. This also means that for $1 \leq i \leq n$ we have that $U_i \in \zeta(\mathtt{a}_i) \cap \zeta(\mathtt{a}'_i)$; the second restriction on ζ then tells us that for $1 \leq i \leq n$ we have that $\mathtt{a}_i = \mathtt{a}'_i$, which entails that $w = w'$. Hence, we conclude that $w = w'$.

We now treat the claims in the order given.

(i) First suppose that $U \in \zeta(L \cap L')$; then there exists a $w \in L \cap L'$ such that $U \in \zeta(w)$. We then have that $U \in \zeta(L)$ and $U \in \zeta(L')$, meaning that $U \in \zeta(L) \cap \zeta(L')$.

For the other inclusion, let $U \in \zeta(L) \cap \zeta(L')$. Then there exist $w \in L$ and $w' \in L'$ such that $U \in \zeta(w)$ and $U \in \zeta(w')$; by (c), we conclude that $w = w'$, and hence $w \in L \cap L'$, which means that $U \in \zeta(L \cap L')$.

(ii) For the third equality, first suppose $U \in \zeta(L \cap L')$. There exists a $w \in L \setminus L'$ such that $U \in \zeta(w)$. This means that $w \in L$ and hence $U \in \zeta(L)$. Furthermore, suppose that $U \in \zeta(L')$; we then find a $w' \in L'$ such that $U \in \zeta(w')$. By (c), this would mean that $w = w'$, and hence $w \in L'$ — a contradiction. We therefore know that $U \notin \zeta(L')$, and hence $U \in \zeta(L) \setminus \zeta(L')$.

For the other inclusion, let $U \in \zeta(L) \setminus \zeta(L')$. Then we obtain $w \in L$ such that $U \in \zeta(L)$, and furthermore there cannot be a $w' \in L'$ with $U \in \zeta(w')$, which in particular means that $w \notin L'$. We conclude that $w \in L \setminus L'$, and hence $U \in \zeta(L \setminus L')$.

(iii) Lastly, if $L = \emptyset$, then $\zeta(L) = \emptyset$ by definition. For the other direction, suppose that $L \neq \emptyset$, i.e., that $w \in L$. By definition of ζ, we have that $\zeta(\mathtt{a}) \neq \emptyset$ for all $\mathtt{a} \in \Sigma$, and hence $\zeta(w)$ cannot be empty, which means that $\zeta(L)$ cannot be empty either. □

8.C Proofs towards correctness of normalisation

The following characterisation of runs labelled by the empty pomset will be useful.

Fact 8.C.1. *Let $\rightsquigarrow_A$ be the smallest relation on Q satisfying the rules*

$$\frac{}{q \rightsquigarrow_A q} \qquad \frac{q \rightsquigarrow_A q'' \qquad q'' \rightsquigarrow_A q'}{q \rightsquigarrow_A q'} \qquad \frac{q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\}) \qquad \forall 1 \leq i \leq n.\ r_i \rightsquigarrow_A r_i' \in F}{q \rightsquigarrow_A q'}$$

Now $q \rightsquigarrow_A q'$ holds if and only if $q \xrightarrow{1}_A q'$.

Proof. The first inclusion is proved by induction on the construction of $\rightsquigarrow_A$. In the base, $q \rightsquigarrow_A q'$ because $q = q'$, thus $q \xrightarrow{1}_A q'$ holds immediately. In the inductive step, there are two cases.

- If $q \rightsquigarrow_A q'$ because there exists a $q'' \in Q$ such that $q \rightsquigarrow_A q''$ and $q'' \rightsquigarrow_A q'$, then by induction we have that $q \xrightarrow{1}_A q''$ and $q'' \xrightarrow{1}_A q'$. It then follows that $q \xrightarrow{1}_A q'$.
- Suppose that $q \rightsquigarrow_A q'$ because there exist $r_1, \ldots, r_n \in Q$ such that $q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\})$ and there exist $r_1', \ldots, r_n' \in F$ such that for $1 \leq i \leq n$ we have that $r_i \xrightarrow{1}_A r_i'$. By induction, we find for $1 \leq i \leq n$ that $r_i \xrightarrow{1}_A r_i'$. We can then conclude that $q \xrightarrow{1}_A q'$.

For the other inclusion, we proceed by induction on the construction of $\rightarrow_A$. In the base, we have $q \xrightarrow{1}_A q'$ because $q = q'$, and so we are done. The case where $p \xrightarrow{1}_A q$ is a sequential unit run cannot occur. For the inductive step, there are two cases to consider.

- If $q \xrightarrow{1}_A q'$ because $1 = U \cdot V$ and there exists a $q'' \in Q$ such that $q \xrightarrow{U}_A q''$ and $q'' \xrightarrow{V}_A q'$, then note that $U = V = 1$. Hence $q \rightsquigarrow_A q''$ and $q'' \rightsquigarrow_A q'$ by induction, meaning $q \rightsquigarrow_A q'$.
- Suppose that $q \xrightarrow{1}_A q'$ because there exist $r_1, \ldots, r_n \in Q$ such that $q' \in \gamma(q, \{\!|r_1, \ldots, r_n|\!\})$ and there exist $r_1' \ldots, r_n' \in F$ such that $1 = U_1 \parallel \cdots \parallel U_n$, and for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r_i'$. By induction, we find for $1 \leq i \leq n$ that $r_i \rightsquigarrow_A r_i'$. We can then conclude that $q \rightsquigarrow_A q'$. □

To prove Lemma 8.33, the following is useful.

Fact 8.C.2. *Let $A = \langle Q, F, \delta, \gamma \rangle$ be a 1-forking automaton without accepting fork targets. For any states $q, q' \in Q$, we have $q \xrightarrow{1}_A q'$ if and only if $q = q'$. Furthermore, $1 \in L_A(q)$ if and only if $q \in F$.*

Proof. For the first part, the implication from right to left holds by definition of $\rightarrow_A$. For the other implication, note that $q \xrightarrow{1}_A q'$ is equivalent to $q \rightsquigarrow_A q'$ by Fact 8.C.1. We proceed by induction on the derivation $q \rightsquigarrow_A q'$. In the base, we have that $q \rightsquigarrow_A q'$ because $q = q'$, and we are done. In the inductive step, there are two cases to consider.

- If $q \rightsquigarrow_A q'$ because $q \rightsquigarrow_A q''$ and $q'' \rightsquigarrow_A q'$ for some $q'' \in Q$, then by induction $q = q'' = q'$.
- If $q \rightsquigarrow_A q'$ because there exist $r_1, \dots, r_n \in Q$ such that $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, and $r'_1, \dots, r'_n \in F$ such that for $1 \leq i \leq n$ we have that $r_i \rightsquigarrow_A r'_i$, then by induction we have for $1 \leq i \leq n$ that $r_i = r'_i$. Since A is 1-forking, it follows that $r_1 = r'_1 \in F$. But this contradicts our premise, because r_1 is a fork target; we can therefore exclude this case.

For the second part, the implication from right to left holds immediately. For the other implication, note that if $1 \in L_A(q)$, then $q \xrightarrow{1}_A q'$ with $q' \in F$. Since $q = q'$ by the first part, $q \in F$. □

Lemma 8.33. *A PA A is well-structured if it is 2-forking, parsimonious, and flat-branching.*

Proof. By definition, an automaton is well-structured if and only if it is 2-forking, flat-branching, and satisfies the property that for all $q \in Q$ and $r \in \phi \in \mathbb{M}(Q)$ with $\gamma(q, \phi) \neq \emptyset$, it holds that $r \notin F$. If the automaton is parsimonious, then this property is satisfied. By Fact 8.C.2, if A is well-structured, then $q \notin F$ is equivalent to $1 \notin L_A(q)$, so it is in particular parsimonious. □

Lemma 8.34. *For any finite pomset automaton A, the predicate $p \xrightarrow{1}_A q$ is decidable.*

Proof. By Fact 8.C.1, it suffices to show that $\rightsquigarrow_A$ is decidable. Since A finite, we can build $\rightsquigarrow_A$ by saturation, starting from the identity and adding pairs until we reach a fixpoint. □

Lemma 8.36. *If a PA A' weakly implements a PA A, then there exists a PA A'' implementing A. Furthermore, if A' is n-forking (respectively flat-branching, parsimonious), then so is A''.*

Proof. Let $A = \langle Q, F, \delta, \gamma \rangle$, and $A' = \langle Q', F', \delta', \gamma' \rangle$. We can assume without loss of generality that Q and Q' are disjoint. We define $A'' = \langle Q'', F'', \delta'', \gamma'' \rangle$ as follows:

$$Q'' = Q' \uplus Q \qquad\qquad F'' = F' \uplus \{q \in Q \;:\; 1 \in L_A(q)\}$$

$$\delta''(q'', \mathtt{a}) = \begin{cases} \delta'(q'', \mathtt{a}) & q'' \in Q' \\ \bigcup_{x \in Q_{q''}} \delta'(x, \mathtt{a}) & q'' \in Q \end{cases} \qquad \gamma''(q'', \phi) = \begin{cases} \gamma'(q'', \phi) & q'' \in Q' \\ \bigcup_{x \in Q_{q''}} \gamma'(x, \phi) & q'' \in Q \end{cases}$$

Here, $Q_{q''}$ is the set of states of Q implementing $q'' \in Q$, by weak implementation. Note that if $q'' \in Q$, then there are finitely many $\phi \in \mathbb{M}(Q'')$ such that $\gamma''(q'', \phi) \neq \emptyset$, because $Q_{q''}$ is finite, and for each $x \in Q_{q''}$ there are finitely many $\phi \in \mathbb{M}(Q')$ such that $\gamma'(x, \phi) \neq \emptyset$.

We now prove that A'' implements A. First, we show that for $q \in Q$ we have $L_A(q) = L_{A''}(q)$.

- For the inclusion from left to right, suppose that $U \in L_A(q)$. If $U = 1$, then $q \in F''$ by definition of F'', and thus $U = 1 \in L_{A''}(q'')$ immediately.

  On the other hand, suppose $U \neq 1$. There exists an $x \in Q_q$ such that $U \in L_{A'}(x)$, because A' weakly implements A. Hence, there must exist a $q' \in F'$ such that $x \xrightarrow{U}_{A'} q'$; a straightforward inductive argument then shows that $x \xrightarrow{U}_{A''} q'$. Now, since $U \neq 1$, the latter run must be non-trivial. We thus find $x' \in Q''$ and $U = V \cdot W$ such that $x \xrightarrow{V}_{A''} x'$ is a unit run, and $x' \xrightarrow{W}_{A''} q'$. By construction of A'', it follows that $q \xrightarrow{V}_{A''} x'$ is a unit run too, and hence $q \xrightarrow{U}_{A''} q'$. Since $q' \in F''$, it follows that $U \in L_{A''}(q)$.

- For the inclusion from right to left, suppose that $U \in L_{A''}(q)$. There must then exist a $q' \in F''$ such that $q \xrightarrow{U}_{A''} q'$. Now, if this run is trivial, then $q = q'$ and $U = 1$. This means that $q \in F''$; since $q \in Q$, it follows that $q \notin F'$, and hence $U = 1 \in L_A(q)$.

  Otherwise, if $q \xrightarrow{U}_{A''} q'$ is non-trivial, then there exists a $q'' \in Q''$ and $U = V \cdot W$ such that $q \xrightarrow{V}_{A''} q''$ is a unit run, and $q'' \xrightarrow{W}_{A''} q'$. By construction of A'', we find an $x \in Q_q$ such that $x \xrightarrow{V}_{A''} q''$ is a unit run, and hence $x \xrightarrow{U}_{A''} q'$. A simple inductive argument then shows that $x \xrightarrow{U}_{A'} q'$ and $q' \in F'$, and thus $U \in L_{A'}(x)$. Because A' weakly implements A, we conclude that $U \in L_A(q)$.

To argue preservation of boundedness and fork-acyclicity, we first observe the following.

(a) For any $q \in Q'$ and $q' \in Q''$, we claim that if $q' \preceq_{A''} q$ then $q' \in Q'$ and $q' \preceq_{A'} q$. To show this, it suffices to argue that if $q' \preceq_{A''} q$ arises from one of the rules that generate $\preceq_{A''}$, then $q' \in Q'$ and $q' \preceq_{A'} q$. This is straightforward, for if $q \in Q'$ then $\delta''(q, \mathtt{a})$ coincides with $\delta'(q, \mathtt{a})$ for all $\mathtt{a} \in \Sigma$, and $\gamma''(q, \phi)$ coincides with $\gamma'(q, \phi)$ for all $\phi \in \mathbb{M}(Q'')$.

(b) For any $q \in Q$ and $q' \in Q''$, we claim that if $q' \preceq_{A''} q$, then $q = q'$, or $q' \in Q'$ such that $q' \preceq_{A'} x$ for some $x \in Q_q$. To see this, we proceed by induction on the construction of $\preceq_{A''}$. In the base, either $q = q'$ by reflexivity, or one of three cases applies.

   - If $q' \preceq_{A''} q$ because $q' \in \delta''(q, \mathtt{a}) \subseteq Q'$ for some $\mathtt{a} \in \Sigma$, then $q' \in \delta'(x, \mathtt{a})$ for some $x \in Q_q$, and therefore $q' \preceq_{A'} x$.
   - If $q' \preceq_{A''} q$ because $q \in \gamma''(q, \phi)$ for some $\phi \in \mathbb{M}(Q'')$, then $q' \in \gamma'(x, \phi) \subseteq Q'$ for some $x \in Q_q$, and therefore $q' \preceq_{A'} x$.
   - If $q' \preceq_{A''} q$ because there exists a $\phi \in \mathbb{M}(Q'')$ such that $q' \in \phi$ and $\gamma''(q, \phi) \neq \emptyset$, then $\phi \in \mathbb{M}(Q')$ and there exists an $x \in Q_q$ such that $\gamma'(x, \phi) \neq \emptyset$, and hence $q' \preceq_{A'} x$.

For the inductive step, $q' \preceq_{A''} q$ because $q'' \in Q''$ and $q' \preceq_{A''} q''$ and $q'' \preceq_{A''} q$. By induction, either $q = q''$ (in which case the claim follows by applying induction to $q' \preceq_{A''} q'' = q$), or $q'' \preceq_{A'} x$ for some $x \in Q_q$, in which case $q' \preceq_{A'} q''$ by (a).

For boundedness, note that by (b) we have that the support of $q \in Q$ in A'' is given by $\{q\} \cup \bigcup_{x \in Q_q} \pi_{A'}(x)$; if A' is bounded, this set must be finite. Furthermore, by (a), we know that the support of $q \in Q'$ in A'' is given by the support of q in A', which is finite if A' is bounded. Thus, if A is bounded, then so is A', and by the above it follows that A'' must also be bounded.

For fork-acyclicity, suppose that A is fork-acyclic; then A' is fork-acyclic, as well. Let $r, q \in Q''$ and $\phi \in \mathbb{M}(Q'')$ with $r \in \phi$ and $\gamma''(q, \phi) \neq \emptyset$. By construction of A'', we have that $r \in Q'$. If $q \preceq_{A''} r$, then by (a) we have that $q \in Q'$ and $q \preceq_{A'} r$. In that case, also $\gamma'(q, \phi) \neq \emptyset$, and hence $r \preceq_{A'} q$. This, however, contradicts the premise that A' is fork-acyclic; we thus have that $r \prec_{A''} q$. We may conclude that A'' must be fork-acyclic as well.

We now prove that our construction preserves n-forking, parsimony and flat-branching.

- If A' is n-forking, let $\phi \in \mathbb{M}(Q'')$ and $q \in Q''$ such that $\gamma''(q, \phi) \neq \emptyset$. By construction there exists $q' \in Q'$ such that $\gamma'(q', \phi) \neq \emptyset$. Since A' is n-forking we may conclude $|\phi| \geq n$.
- Assume, towards a contradiction, that A'' is not parsimonious. Then there exist $q \in Q''$ and $r \in \phi \in \mathbb{M}(Q'')$ such that $\gamma''(q, \phi) \neq \emptyset$ but $1 \in L_{A''}(r)$. By construction there exists $q' \in Q'$ such that $\gamma'(q', \phi) \neq \emptyset$. We also know that it must be the case that $r \in Q'$, hence $1 \in L_{A'}(r)$. This contradicts the premise that A' is parsimonious.
- Assume, towards a contradiction, that A is not flat-branching. There must then exist $q \in Q''$ and $\phi, \psi \in \mathbb{M}(Q'')$ with $r \in \phi$ such that $\gamma''(q, \phi) \neq \emptyset$ and $\gamma''(r, \psi) \cap F \neq \emptyset$. By construction of A'', we find that $\phi \in \mathbb{M}(Q')$, and there exists a $q' \in Q'$ such that $\gamma'(q', \phi) \neq \emptyset$. Also by construction of A'' and the fact that $r \in \phi$ (and hence $r \in Q'$), we find that $\psi \in \mathbb{M}(Q')$ and $\gamma'(r, \psi) \cap F' \neq \emptyset$. This contradicts the premise that A' is flat-branching. □

8.C.1 Proof of correctness for parsimonification

Lemma 8.39. *A_0 is parsimonious and weakly implements A.*

Proof. We relate runs in A to runs in A_0 and vice versa, as follows:

Fact 8.C.3. *If $q \xrightarrow{U}_A q'$, then $q \xrightarrow{U}_{A_0} q'$. If also $1 \in L_A(q')$ and $U \neq 1$, then $q \xrightarrow{U}_{A_0} \top$.*

Proof of Fact 8.C.3. We proceed by induction on $q \xrightarrow{U}_A q'$. In the base, there are two cases. On the one hand, if $q \xrightarrow{U}_A q'$ because $q = q'$ and $U = 1$, then $q \xrightarrow{U}_{A_0} q'$; the second claim holds vacuously.

On the other hand, if $q \xrightarrow{U}_A q'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q' \in \delta(q, \mathtt{a})$, then $q' \in \delta_0(q, \mathtt{a})$ by construction, so indeed $q \xrightarrow{U}_{A_0} q'$. If $1 \in L_A(q')$, then $\top \in \delta_0(q, \mathtt{a})$ as well, so $q \xrightarrow{\mathtt{a}}_{A_0} \top$.

For the inductive step, we have two more cases.

- If $q \xrightarrow{U}_A q'$ because $U = V \cdot W$ and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$, then by induction we know that $q \xrightarrow{V}_{A_0} q''$ and $q'' \xrightarrow{W}_{A_0} q'$, so $q \xrightarrow{U}_{A_0} q''$.

  If furthermore $1 \in L_A(q')$ and $V \cdot W \neq 1$, then we distinguish two subcases.

  - If $W = 1$, then $V = U$, and $1 \in L_A(q'')$; therefore by induction we get $q \xrightarrow{U}_{A_0} \top$.
  - If $W \neq 1$, then by induction we get $q'' \xrightarrow{W}_{A_0} \top$, hence $q \xrightarrow{V \cdot W}_{A_0} \top$.

- Suppose that $q \xrightarrow{U}_A q'$ because there exist $r_1, \dots, r_n \in Q$ with $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, and there exist $r'_1, \dots, r'_n \in F$ such that $U = U_1 \parallel \cdots \parallel U_n$ and for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r'_i$. We then partition $\{\!|r_1, \dots, r_n|\!\}$ into ϕ and ψ such that for all $1 \leq i \leq n$ we have that $r_i \in \phi$ implies $U_i \neq 1$, and $r_i \in \psi$ implies $U_i = 1$. By construction of γ_0, we have $q \in \gamma_0(q, \phi)$. By induction, since for all $1 \leq i \leq n$ with $r_i \in \phi$ we have $U_i \neq 1$, we obtain that for all $1 \leq i \leq n$ with $r_i \in \phi$ we have $r_i \xrightarrow{U_i}_{A_0} \top$. We may therefore conclude that $q \xrightarrow{U_1 \parallel \cdots \parallel U_n}_{A_0} q'$.

  If additionally $1 \in L_A(q)$ and $U_1 \parallel \cdots \parallel U_n \neq 1$, then there must exist a $1 \leq i \leq n$ with $U_i \neq 1$, so we know that $\phi \neq \boxslash$; hence, $\top \in \gamma_0(q, \phi)$. As result, we get $q \xrightarrow{U_1 \parallel \cdots \parallel U_n}_{A_0} \top$. □

Fact 8.C.4. *If $q \in Q$ and $q \xrightarrow{U}_{A_0} q'$, then the following hold:*

(i) If $q' \in Q$, then $q \xrightarrow{U}_A q'$.

(ii) If $q' = \top$, then $U \in L_A(q)$ but $U \neq 1$.

Proof of Fact 8.C.4. In the base, there are two cases.

- If $q \xrightarrow{U}_{A_0} q'$ because $U = 1$ and $q = q'$, then $q \xrightarrow{U}_A q'$ immediately.
- If $q \xrightarrow{U}_{A_0} q'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q' \in \delta_0(q, \mathtt{a})$, then there are two subcases:

  (i) If $q' \in Q$, then $q' \in \delta(q, \mathtt{a})$, and hence $q \xrightarrow{U}_A q'$.

  (ii) If $q' = \top$, then there exists a $q'' \in \delta(q, \mathtt{a})$ such that $1 \in L_A(q'')$. In particular, this means that $q \xrightarrow{U}_A q''$, whence $U \in L_A(q)$. Since $\mathtt{a} \neq 1$, we rightfully get $U \neq 1$.

For the inductive step, there are again two cases.

- Suppose $q \xrightarrow{U}_{A_0} q'$ because $U = V \cdot W$ and there exists a $q'' \in Q_0$ with $q \xrightarrow{V}_{A_0} q''$ and $q'' \xrightarrow{W}_{A_0} q'$. We distinguish two cases based on q'':

(i) If $q'' = \top$, then by construction of A_0 we have $W = 1$ and $q' = \top$, so by induction it follows that $U = V \in L_A(q)$ while $U = V \neq 1$;

(ii) If $q'' \in Q$, then by induction we have $q \xrightarrow{U}_A q''$. We distinguish two cases:

- If $q' \in Q$, then by induction $q'' \xrightarrow{V}_A q'$, so $q \xrightarrow{U \cdot V}_A q'$.
- If $q' = \top$, then by induction $W \in L_A(r)$ while $W \neq 1$, so $U \in L_A(q)$ while $U \neq 1$.

- Suppose $q \xrightarrow{U}_{A_0} q'$ because there exist $r_1, \dots, r_n \in Q_0$ such that $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$, and $U = U_1 \parallel \cdots \parallel U_n$ such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_{A_0} \top$. By construction of γ_0, we also know that for $1 \leq i \leq n$ we have $r_i \in Q$. By induction, we then know that for $1 \leq i \leq n$ we have $r_i' \in F$ such that $r_i \xrightarrow{U_i}_A r_i'$ but $U_i \neq 1$. We distinguish the two cases for q':

(i) If $q' \in Q$, then by construction of γ_0 there exists $\psi \in \mathbb{M}(Q)$ such that $\psi = \{\!|r_{n+1}, \dots, r_m|\!\}$ and $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\} \sqcup \psi)$, and furthermore for all $n < i \leq m$ there exists an $r_i' \in F$ such that $r_i \xrightarrow{1}_A r_i'$. We may then complete the run by choosing for $n < i \leq m$ that $U_i = 1$ and $r_i' = r_i$, which allows us to conclude that $q \xrightarrow{U_1 \parallel \cdots \parallel U_n \parallel U_{n+1} \parallel \cdots \parallel U_m} q'$. Since $U_{n+1} \parallel \cdots \parallel U_m = 1 \parallel \cdots \parallel 1 = 1$, we are done.

(ii) If $q = \top$, then $n > 0$ and there exist $\psi \in \mathbb{M}(Q)$ and $q'' \in Q$ such that $\psi = \{\!|r_{n+1}, \dots, r_m|\!\}$ and $q'' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\} \sqcup \psi)$, and $1 \in L_A(q'')$. As in the previous case, we find $q \xrightarrow{U_1 \parallel \cdots \parallel U_n}_A q''$. Since $1 \in L_A(q'')$, this implies $U_1 \parallel \cdots \parallel U_n \in L_A(q)$. To conclude, note that for $1 \leq i \leq n$ we have $U_i \neq 1$; since $n > 0$, we have $U = U_1 \parallel \cdots \parallel U_n \neq 1$. □

Together, the above facts imply that $L_A(q) \setminus \{1\} = L_{A_0}(q)$. So, if $1 \notin L_A(q)$, then we have $L_A(q) = L_{A_0}(q)$, and otherwise $L_A(q) = L_{A_0}(q) \cup L_{A_0}(\top)$. Therefore, A_0 weakly implements A, since boundedness and fork-acyclicity are clearly preserved by this construction.

To show that A_0 is parsimonious, notice that by construction, if $\gamma_0(q, \phi) \neq \emptyset$ and $r \in \phi$, then $r \in Q$. As we showed above, this implies $1 \notin L_{A_0}(r)$. □

8.C.2 Proof of correctness for nullary fork elimination

Lemma 8.42. *A_1 is 1-forking, parsimonious, and weakly implements A.*

Proof. The condition $\phi \neq \boxslash$ in the definition of γ_1 ensures that A_1 is 1-forking. Before we show that A_1 is parsimonious, note that if we have $r \in \phi$ and $\gamma_1(q, \phi) \neq \emptyset$, then there is a state q' such that $q \xrightarrow{1}_A q'$ and $\gamma(q', \phi) \neq \emptyset$. By parsimony of A, we have $1 \notin L_A(r)$, hence $r \notin F$. By Fact 8.C.2, we conclude that $1 \notin L_{A_1}(p)$, hence A_1 is parsimonious.

We now check that A_1 weakly implements A, by relating their runs.

Fact 8.C.5. *If $q \xrightarrow{1}_A p \xrightarrow{U}_A p' \xrightarrow{1}_A q'$ with $U \neq 1$, then $q \xrightarrow{U}_{A_1} q'$.*

Proof of Fact 8.C.5. We proceed by induction on the construction of $p' \xrightarrow{U}_A q'$. In the base, we can exclude the case where $p' = q'$ and $U = 1$, because it contradicts the premise. This leaves the case where $p \xrightarrow{U}_A p'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $p' \in \delta(p, \mathtt{a})$. By construction of δ_1, we then find that $q' \in \delta_1(q, \mathtt{a})$, hence $q \xrightarrow{U}_{A_1} q'$. For the inductive step, there are two cases:

- Suppose $p \xrightarrow{U}_A p'$ because $U = V \cdot W$ and there exists $p'' \in Q$ such that $p \xrightarrow{V}_A p''$ and $p'' \xrightarrow{W}_A p'$. We distinguish four subcases:
  - If $V = 1 = W$, then $U = V \cdot W = 1$, contradicting the premise; we disregard this case.
  - If $V \neq 1 = W$, then $p'' \xrightarrow{1}_A q'$ and $U = V$; by induction, we find that $q \xrightarrow{U}_{A_1} q'$.
  - If $V = 1 \neq W$, then $q \xrightarrow{1}_A p''$ and $U = W$; by induction, we find that $q \xrightarrow{U}_{A_1} q'$.
  - If $V \neq 1 \neq W$, then induction we get $q \xrightarrow{V}_{A_1} p''$ and $p'' \xrightarrow{W}_{A_1} q'$, hence $q \xrightarrow{U}_{A_1} q'$.
- Suppose $p \xrightarrow{U}_A p$ because there exist $r_1, \ldots, r_n \in Q$ such that $p' \in \gamma(p, \{\!|r_1, \ldots, r_n|\!\})$, and $U = U_1 \parallel \cdots \parallel U_n$ such that for $1 \leq i \leq n$ there exists $r_i' \in F$ with $r_i \xrightarrow{U_1}_A r_i'$. By parsimony of A, we know that each of the U_i is different from 1. Hence, $U_1 \parallel \cdots \parallel U_n$ is itself non-empty, and we have $n > 0$. Therefore $q' \in \gamma_1(q, \{\!|r_1, \ldots, r_n|\!\})$. By induction we have for every $1 \leq i \leq n$ that $r_i \xrightarrow{U_i}_{A_1} r_i' \in F$, so we may conclude $q \xrightarrow{U}_{A_1} q'$. □

Fact 8.C.6. *If $q \xrightarrow{U}_{A_1} q'$, then $q \xrightarrow{U}_A q'$.*

Proof of Fact 8.C.6. We proceed by induction on $q \xrightarrow{U}_{A_1} q'$. In the base, there are two cases.

- If $q \xrightarrow{U}_{A_1} q'$ because $q = q'$ and $U = 1$, then $q \xrightarrow{U}_A q'$ immediately.
- If $q \xrightarrow{U}_{A_1} q'$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q' \in \delta_1(q, \mathtt{a})$, then there exist $p, p' \in Q$ such that $q \xrightarrow{1}_A p$ and $p' \xrightarrow{1}_A q'$ with $p' \in \delta(p, \mathtt{a})$, by construction of δ_1. We can string these together to find that $q \xrightarrow{1}_A p \xrightarrow{\mathtt{a}}_A p' \xrightarrow{1}_A q'$, hence $q \xrightarrow{U}_A q'$.

In the inductive step, there are again two cases:

- Suppose $q \xrightarrow{U}_{A_1} q'$ because $U = V \cdot W$ and there exists a $q'' \in Q$ such that $q \xrightarrow{V}_{A_1} q''$ and $q'' \xrightarrow{W}_{A_1} q'$. By induction, $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$, and hence $q \xrightarrow{U}_A q'$.
- Suppose $q \xrightarrow{U}_{A_1} q'$ because there exist $r_1, \ldots, r_n \in Q$ with $q' \in \gamma_1(q, \{\!|r_1, \ldots, r_n|\!\})$, and $U = U_1 \parallel \cdots \parallel U_n$ and for $1 \leq i \leq n$ there exists $r_i' \in F$ with $r_i \xrightarrow{U_i}_{A_1} r_i'$. By induction, it must be the case that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r_i'$. By construction of γ_1, we know that

there are $p, p' \in Q$ such that $q \xrightarrow{1}_A p$, $p' \xrightarrow{1}_A q'$, and $p' \in \gamma(p, \{\!|q_1, \dots, q_n|\!\})$. We then know that $p \xrightarrow{U_1 \| \cdots \| U_n}_A p'$, and hence $q \xrightarrow{U_1 \| \cdots \| U_n}_A q'$. □

We can now wrap up by showing that, for $q \in Q$, we have $L_A(q) = \bigcup_{q \xrightarrow{1}_A p} L_{A_1}(p)$.

- Let $U \in L_A(q)$, meaning there is $q' \in F$ such that $q \xrightarrow{U}_A q'$. On the one hand, if $U = 1$, then $1 \in L_{A_1}(q')$; since $q \xrightarrow{1}_A q'$, U is contained in the right-hand side. On the other hand, if $U \neq 1$, then $q \xrightarrow{U}_{A_1} q'$ by Fact 8.C.5, hence $U \in L_{A_1}(p)$. Since $q \xrightarrow{1}_A q$, we are done.
- Let $p \in Q$ such that $q \xrightarrow{1}_A p$, and let $U \in L_{A_1}(p)$. This means there is $p' \in F$ such that $p \xrightarrow{U}_{A_1} p'$, and hence $p \xrightarrow{U}_A p'$ by Fact 8.C.6. Therefore $q \xrightarrow{U}_A p' \in F$, so $U \in L_A(q)$.

Since boundedness and fork-acyclicity are preserved, A_1 weakly implements A. □

8.C.3 Proof of correctness for unary fork elimination

Lemma 8.47. *A_2 is parsimonious, 2-forking, and implements A.*

Proof. We start by showing that our construction preserves fork-acyclicity and boundedness. For preservation of fork-acyclicity, we first verify the following.

Fact 8.C.7. *The following hold for all $w = q_1 \cdots q_n \in Q_2$ and $w' = q'_1 \cdots q'_m \in Q_2$:*

(i) If $q \in Q$ such that $q \uparrow w$, then for all q_i we have $q_i \preceq_A q$.

(ii) If $w' \in \delta_2(w, \mathtt{a})$, then for every q'_i there exists a q_j with $q'_i \preceq_A q_j$.

(iii) If $w' \in \gamma_2(w, \phi)$, then for every q'_i there exists a q_j with $q'_i \preceq_A q_j$.

(iv) If $\gamma_2(w, \phi) \neq \emptyset$ with $r \in \phi$, then there exists a q_i with $r \prec_A q_i$.

(v) If $w' \preceq_{A_2} w$, then for every q'_i there exists a q_j with $q'_i \preceq_A q_j$.

Proof of Fact 8.C.7. We treat the claims in the order given.

(i) We proceed by induction on the construction of $\uparrow$. In the base, where $q = q_1$, the claim holds vacuously. For the inductive step, we have $r \in Q$ such that $r \uparrow q_1 \cdots q_{n-1}$ and $q_n \in \gamma(q, \{\!|r|\!\})$. In that case, $q_n \preceq_A q$ and $r \preceq_A q$ immediately. Also, we find by induction that for all $1 \leq i < n$ it holds that $q_i \preceq_A r$, and hence $q_i \preceq_A q$.

(ii) We proceed by induction on the construction of δ_2. In the base, we have $w' \in \delta_2(w, \mathtt{a})$ because $w' = q' \cdot x \cdot y$ and $w = q \cdot y$, and there exists an $r \in Q$ such that $q \uparrow r \cdot x$ and $q' \in \delta(r, \mathtt{a})$. By (i), we know that $q'_1 = q' \preceq_A r \preceq_A q = q_1$. Furthermore, if q'_i appears in x, then also by (i) we know that $q'_i \preceq_A q = q_1$. Lastly, if q'_i appears in y, then note that it also appears in w, and hence we can conclude by $q'_i \preceq_A q_j$ for some j. In the inductive step, $w = q \cdot x$ such that $q \in F$ and $w' \in \delta_2(x, \mathtt{a})$. The claim then follows immediately by induction.

(iii) This proof proceeds analogously to the one above.

(iv) We proceed by induction on the construction of γ_2. In the base, we have that $w = q_1 \cdot x$ and $w' = q'_1 \cdot y \cdot x$ such that there exists an $p \in Q$ with $q_1 \uparrow p \cdot y$ and $q'_1 \in \gamma(p, \phi)$. Since A is fork-acyclic, it follows that $r \prec_A p$; because $p \preceq_A q_1$ by (i), the claim follows.

For the inductive step, we have that $\gamma_2(w, \phi) \neq \emptyset$ because $w = q \cdot x$ such that $q \in F$ and $\gamma_2(x, \phi) \neq \emptyset$. By induction, we then find a q_i such that $r \prec_A q_i$.

(v) This can be shown by induction on $\preceq_{A_2}$, noting that the base cases are covered by (ii)–(iv). For the inductive step, it suffices to note that the claimed property is transitive in nature. □

Now, if $w \in \phi \in \mathbb{M}(Q_2)$ and $x \in Q_2$ with $\gamma_2(x, \phi) \neq \emptyset$, we should show that $w \prec_{A_2} w$. First, note that $w = r \in Q$ for some $r \in Q$ by construction of γ_2. By Fact 8.C.7(iv), we know that $w = q_1 \cdots q_n$, and there exists a $1 \leq i \leq n$ with $r \prec_A q_i$. Suppose, towards a contradiction, that $w \preceq_{A_2} r$; then by Fact 8.C.7(v) we also know that $q_i \preceq_A r$, contradicting that $r \prec_A q_i$.

To argue that A_2 is bounded, we first record the following.

Fact 8.C.8. *Let $w = q_1 \cdots q_n \in Q_2$ and $x = q'_1 \cdots q_{n'} \in Q_2$. If $w \preceq_{A_2} x$, then for every $1 \leq i \leq n$ there exists a $1 \leq j \leq n'$ such that $n + D_A(q_i) - i \leq n' + D_A(q'_j) - j$.*

Proof of Fact 8.C.8. It suffices to verify the claim for the pairs generating $\preceq_{A_2}$.

- If $w \in \delta_2(x, \mathtt{a})$ for some $\mathtt{a} \in \Sigma$, then we proceed by induction on the construction of δ_2. In the base, there exist $r \in Q$ and $1 \leq k \leq n$ such that $q'_1 \uparrow r \cdot q_2 \cdots q_k$ and $q_1 \in \delta(r, \mathtt{a})$, while $n' + k - 1 = n$, and for $k < i \leq n'$ we have $q_i = q'_{i-k+1}$. We now consider two cases.
    - When $1 \leq i \leq k$, we choose $j = 1$; since $n \leq n'$ and $i \geq 1$ and $D_A(q_i) \leq D_A(q'_1)$ by Fact 8.C.7(i), we find that $n + D_A(q_i) - i \leq n' + D_A(q'_1) - 1$.
    - Otherwise, when $k < i \leq n$, we choose $j = i - k + 1$ to find that $n + D_A(q_i) - i = n + D_A(q'_j) - i = n' + k - 1 - D_A(q'_j) - i = n' + D_A(q'_j) - j$

  In the inductive step, $q'_1 \in F$ and $w \in \delta_2(q'_2 \cdots q'_{n'}, \mathtt{a})$. The claim then follows by induction.
- If $w \in \gamma_2(x, \phi)$ for some $\phi \in \mathbb{M}(Q_2)$, then the proof is similar to the previous case.

- If there exists a $\phi \in \mathbb{M}(Q_2)$ such that $w \in \phi$ and $\gamma_2(x, \phi) \neq \emptyset$, then note that $\phi \in \mathbb{M}(Q)$ by construction of γ_2, and thus that $w = r$ for some $r \in Q$. The proof proceeds by induction on γ_2, where it suffices to show that $D_A(r) < D_A(q'_j)$ for some $1 \leq j \leq n'$. This is a direct consequence of Fact 8.C.7(iv). □

Now, suppose that A is bounded. To see that A_2 is bounded, let $q_1 \cdots q_n \in Q_2$ and choose $m = \max_{1 \leq i \leq n} D_A(q_i)$. If $q'_1, \dots, q'_{n'} \in Q$ such that $q'_1 \cdots q_{n'} \preceq_A q_1 \cdots q_n$, then by Fact 8.C.8, we find $1 \leq j \leq n'$ such that $n' \leq n' + D_A(q'_1) - 1 \leq n + D_A(q_j) - j \leq n + m$. By Fact 8.C.7(v), $q'_1, \dots, q'_n \in \pi_A(q_1) \cup \cdots \cup \pi_A(q_n)$; the latter set is finite. Hence, the states supporting $q_1 \cdots q_n$ in A_2 are words of length at most $n + m$ over a finite alphabet; thus $\pi_{A_2}(q_1 \cdots q_n)$ must be finite.

To show that A_2 can accept the same languages as accepted by A, the following facts are useful.

Fact 8.C.9. *If $q \in Q$ and $q \uparrow q_1 \cdots q_n$ such that for $1 \leq i \leq n$ there exist $q'_i \in Q$ and $U_i \in \mathsf{SP}$ with $q_i \xrightarrow{U_i}_A q'_i$, and for $1 \leq i < n$ it holds that $q'_i \in F$, then $q \xrightarrow{U_1 \cdots U_n}_A q'_n$.*

Proof of Fact 8.C.9. The proof proceeds by induction on the construction of $\uparrow$. In the base, we have that $q = q_1$ and $n = 1$; it then follows immediately that $q = q_1 \xrightarrow{U_1}_A q'_1 = q'_n$.

For the inductive step, we have $q \uparrow q_1 \cdots q_n$ because there exists $r \in Q$ with $r \uparrow q_1 \cdots q_{n-1}$ and $q_n \in \gamma(q, \{\!|r|\!\})$. By induction, we then know that $r \xrightarrow{U_1 \cdots U_{n-1}}_A q_{n-1}$; since $q_{n-1} \in F$ and $q_n \in \gamma(q, \{\!|r|\!\})$ it follows that $q \xrightarrow{U_1 \cdots U_{n-1}}_A q_n \xrightarrow{U_n} q'_n$, and hence $q \xrightarrow{U_1 \cdots U_n} q'_n$. □

Fact 8.C.10. *If $w \in Q_2$ and $w' \in F_2$ and $U \in \mathsf{SP}$ such that $w \xrightarrow{U}_{A_2} w'$, then $w = q_1 \cdots q_n$ and $U = U_1 \cdots U_n$ such that for $1 \leq i \leq n$ there exists a $q'_i \in F$ with $q_i \xrightarrow{U_i}_A q'_i$.*

Proof of Fact 8.C.10. We proceed by induction on the length ℓ of $w \xrightarrow{U}_{A_2} w'$. In the base, where $\ell = 0$, we have $U = 1$ and $w = w' \in F_2$. We choose for $1 \leq i \leq n$ that $U_i = 1$ and $q'_i = q_i \in F$.

For the inductive step, we have that $U = V \cdot W$ and $w'' \in Q_2$ such that $w \xrightarrow{V}_{A_2} w''$ is a unit run, and $w'' \xrightarrow{W}_{A_2} w'$ of length $\ell - 1$. By induction, $w'' = r_1 \cdots r_m$ and $W = W_1 \cdots W_m$ such that for $1 \leq i \leq m$ there exists an $r'_i \in F$ with $r_i \xrightarrow{W_i}_A r'_i$. Suppose that $w \xrightarrow{V}_{A_2} w''$ is a sequential unit run. Then $V = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $w'' \in \delta_2(w, \mathtt{a})$. We proceed by induction on δ_2.

In the base, we have that $w = q_1 \cdots q_n$ and $w'' = q' \cdot v \cdot q_2 \cdots q_n$ such that there exists an $r \in Q$ with $q_1 \uparrow r \cdot v$ and $q' \in \delta(r, \mathtt{a})$. Note that $q' \cdot v = r_1 \cdots r_k$ for some $k \leq m$, and that $r_{i+k-1} = q_i$ for $2 \leq i \leq n$. We choose $U_1 = \mathtt{a} \cdot W_1 \cdots W_k$. Since $r \xrightarrow{\mathtt{a} \cdot W_1}_A r'_1 \in F$ and for $2 \leq i \leq k$ we have $r_i \xrightarrow{W_i}_A r'_i \in F$, it follows that $q_1 \xrightarrow{U_1}_A r'_k$ by Fact 8.C.9; we set $q'_1 = r'_k$. For $i \geq 2$, we choose $q'_i = r'_{i+k-1}$ and $U_i = W_{i+k-1}$, to find that $r_{i+k-1} \xrightarrow{W_{i+k-1}}_A r'_{i+k-1}$, and hence $q_i \xrightarrow{U_i} q'_i$. Finally, we note that $U_1 \cdots U_n = \mathtt{a} \cdot W_1 \cdots W_k \cdot W_{k+1} \cdots W_m = V \cdot W = U$.

In the inductive step, $w = q_1 \cdots q_n$ and $w'' \in \delta_2(q_2 \cdots q_n, \mathtt{a})$, with $q_1 \in F$. By induction, $U = U_2 \cdots U_n$ where, for $2 \leq i \leq n$, $q_i' \in F$ with $q_i \xrightarrow{U_i}_A q_i'$. Here, $U_1 = 1$ and $q_1' = q_1$ suffices.

The case where $w \xrightarrow{V}_{A_2} w''$ is a parallel unit run can be treated similarly. □

Fact 8.C.10 tells us that for $q \in Q$ we have that $L_{A_2}(q) \subseteq L_A(q)$. After all, if $q \xrightarrow{U}_{A_2} w$ for some $w \in F_2$, then we find $q' \in F$ such that $q \xrightarrow{U}_A q'$, and hence $U \in L_A(q)$.

For the converse inclusion, the following facts tell us how we can compose runs in A_2.

Fact 8.C.11. *If $w \xrightarrow{U}_{A_2} w'$ and $x \in Q_2$, then $w \cdot x \xrightarrow{U}_{A_2} w' \cdot x$.*

Proof of Fact 8.C.11. We proceed by induction on the length ℓ of $w \xrightarrow{U}_{A_2} w'$. In the base, where $\ell = 0$, we have that $w = w'$ and $U = 1$. We then know that $w \cdot x = w' \cdot x$; hence $w \cdot x \xrightarrow{U}_{A_2} w' \cdot x$.

For the inductive step, let $\ell > 1$. We then find $w'' \in Q_2$ and $U = V \cdot W$ such that $w \xrightarrow{V}_{A_2} w''$ is a unit run, and $w'' \xrightarrow{W}_{A_2} w'$ is of length $\ell - 1$. Hence, $w'' \cdot x \xrightarrow{W}_{A_2} w' \cdot x$ by induction. If $w \xrightarrow{V}_{A_2} w''$ is a sequential unit run, then $V = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $w'' \in \delta_2(w, \mathtt{a})$. By construction of δ_2, we have $w'' \cdot x \in \delta_2(w \cdot x, \mathtt{a})$, which means that $w \cdot x \xrightarrow{V}_{A_2} w'' \cdot x$. In total, we have $w \cdot x \xrightarrow{U}_{A_2} w' \cdot x$.

The case where $w \xrightarrow{V}_{A_2} w''$ is a parallel unit run can be treated similarly. □

Fact 8.C.12. *Let $q, r \in Q$ with $q' \in \gamma(q, \{\!|r|\!\})$, and let $r \xrightarrow{U}_{A_2} w$ be nontrivial. Then $q \xrightarrow{U}_{A_2} w \cdot q'$.*

Proof of Fact 8.C.12. Since $r \xrightarrow{U}_{A_2} w$ is nontrivial, $U = V \cdot W$ and $x \in Q_2$ such that $r \xrightarrow{V}_{A_2} x$ is a unit run, and $x \xrightarrow{W}_{A_2} w$. If $r \xrightarrow{V}_{A_2} x$ is a sequential unit run, then $V = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $x \in \delta_2(r, \mathtt{a})$. By construction of δ_2, we obtain $q'', r \in Q$ and $y \in Q_2$ such that $x = q'' \cdot y$ and $r \uparrow r' \cdot y$ as well as $q'' \in \delta(r', \mathtt{a})$. By definition of $\uparrow$, also $q \uparrow r' \cdot y \cdot q'$, and thus $x \cdot q' = q'' \cdot y \cdot q' \in \delta_2(q, \mathtt{a})$. We then find that $q \xrightarrow{V}_{A_2} x \cdot q'$. Since $x \cdot q' \xrightarrow{W}_{A_2} w \cdot q'$ by Fact 8.C.11, we conclude that $q \xrightarrow{U}_{A_2} w \cdot q'$.

The case where $r \xrightarrow{V}_{A_2}$ is a parallel unit run can be argued similarly. □

Fact 8.C.13. *Let $w \xrightarrow{U}_{A_2} w'$ be nontrivial, and $x \in F_2$. Then $x \cdot w \xrightarrow{U}_{A_2} w'$.*

Proof of Fact 8.C.13. Since $w \xrightarrow{U}_{A_2} w'$ is non-trivial, we find that $U = V \cdot W$ and $w'' \in Q_2$ such that $w \xrightarrow{V}_{A_2} w''$ is a unit run, and $w'' \xrightarrow{W}_{A_2} w'$. If $w \xrightarrow{V}_{A_2} w''$ is a sequential unit run, then $V = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $w'' \in \delta_2(w, \mathtt{a})$. A simple inductive argument on the length of x then tells us that $w'' \in \delta_2(x \cdot w, \mathtt{a})$ as well. From this, it follows that $x \cdot w \xrightarrow{V}_{A_2} w''$, and thus $x \cdot w \xrightarrow{U}_{A_2} w'$.

The case where $w \xrightarrow{V}_{A_2} w''$ is a parallel unit run can be argued similarly. □

Finally, we can use the above to show that A_2 can simulate the unary forks of A.

Fact 8.C.14. *If $q \xrightarrow{U}_A q'$, then there exists $x \in F_2$ such that $q \xrightarrow{U}_{A_2} x \cdot q'$.*

Proof of Fact 8.C.14. We proceed by induction on $q \xrightarrow{U}_A q'$. If $q \xrightarrow{U}_A q'$ is trivial, then the claim is satisfied by choosing $x = 1$. Otherwise, suppose $q \xrightarrow{U}_A q'$ is a sequential unit run, i.e., $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, and $q' \in \delta(q, \mathtt{a})$. Since $q \uparrow q$, we then have that $q' \in \delta_2(q, \mathtt{a})$, and hence $q \xrightarrow{U}_{A_2} q'$.

For the inductive step, there are again two cases to consider.

- Suppose $q \xrightarrow{U}_A q'$ because $U = V \cdot W$ and there exists $q'' \in Q$ such that $q \xrightarrow{V}_A q''$ and $q'' \xrightarrow{W}_A q'$. By induction, we obtain $x'', x' \in F_2$ such that $q \xrightarrow{V}_{A_2} x'' \cdot q''$ and $q'' \xrightarrow{W}_{A_2} x' \cdot q'$. Without loss of generality, $q'' \xrightarrow{W}_A q'$ is non-trivial, and hence neither is $q'' \xrightarrow{W}_{A_2} x' \cdot q'$. By Fact 8.C.13, we find that $x'' \cdot q'' \xrightarrow{W}_{A_2} x' \cdot q'$. In total, we find that $q \xrightarrow{U}_{A_2} x \cdot q'$.
- Suppose $q \xrightarrow{U}_A q'$ because $U = U_1 \parallel \cdots \parallel U_n$, and there exist $r_1, \dots, r_n \in Q$ and $r'_1, \dots, r'_n \in F$ such that for $1 \leq i \leq n$ we have $r_i \xrightarrow{U_i}_A r'_i$, and $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$. There are two subcases.
  - If $n = 1$, then by induction we find $x_1 \in F_2$ such that $r_1 \xrightarrow{U_1}_{A_2} x'_1 \cdot r'_1$. By Fact 8.C.12, we then find $q \xrightarrow{U}_{A_2} x'_1 \cdot r'_1 \cdot q'$. Choosing $x' = x'_1 \cdot r'_1$ satisfies the claim.
  - If $n \geq 2$, then $q' \in \gamma_2(q, \{\!|r_1, \dots, r_n|\!\})$. By induction, we find for $1 \leq i \leq n$ an $x_i \in F_2$ with $r_i \xrightarrow{U_i}_{A_2} x'_i \cdot r'_i \in F_2$. Thus $q \xrightarrow{U}_{A_2} q'$; choosing $x' = 1$ satisfies the claim. □

The above allows us to prove that, for $q \in Q$, we have $L_A(q) \subseteq L_{A_2}(q)$. To this end, suppose $U \in L_A(q)$; then there exists a $q' \in F$ with $q \xrightarrow{U}_A q'$. By Fact 8.C.14 we find $x' \in F_2$ with $q \xrightarrow{U}_{A_2} x \cdot q' \in F_2$, and hence $U \in L_{A_2}(q)$. Since $L_{A_2}(q) \subseteq L_A(q)$, it follows that $L_A(q) = L_{A_2}(q)$.

Note that it is 2-forking by construction. For parsimony, observe that if $w \in \phi \in \mathbb{M}(Q_2)$ and $x \in Q_2$ such that $\gamma_2(x, \phi)$, then $\phi \in \mathbb{M}(Q)$ by definition of γ_2, and hence $w = q$ for some $q \in Q$. A simple inductive argument then tells us that there exists an $r \in Q$ such that $\gamma(r, \phi) \neq \emptyset$. Since A is parsimonious, we know that $1 \notin L_A(q)$; since $L_A(q) = L_{A_2}(q)$, it follows that $1 \notin L_{A_2}(q)$. □

8.C.4 Proof of correctness for flat-branching

Lemma 8.52. *A_3 is 2-forking, parsimonious, flat-branching, and weakly implements A.*

Proof. The proof of this statement consists of several steps. In the sequel, we will write $Q^{\mathbf{p}} = \{q^{\mathbf{p}} : q \in Q\}$ and $Q^{\mathbf{s}} = \{q^{\mathbf{s}} : q \in Q\}$. We start by making the following observations:

Fact 8.C.15. *The following hold for all $\psi \in \mathbb{M}(Q)$:*

(i) If $\phi \blacktriangleleft \psi$, then $|\phi| \leq |\psi|$.

(ii) If $p \in Q_3$ and $\phi \in \mathbb{M}(Q_3)$ such that $\gamma_3(p, \phi) \neq \emptyset$, then $\phi \in \mathbb{M}(Q^{\mathbf{s}})$.

(iii) If $p \in Q$ and $\phi \in \mathbb{M}(Q_3)$, then $\gamma_3(p^{\mathbf{s}}, \phi) \subseteq Q^{\mathbf{s}} \cup Q^{\mathbf{p}}$.

Proof. We treat the claims in the order given.

(i) This claim is proved by induction on $\blacktriangleleft$. In the base, $\phi \blacktriangleleft \psi$ because $\phi = \psi$, and so the claim holds immediately. For the inductive step, we have that $\phi \blacktriangleleft \psi$ because $\psi = \psi_1 \sqcup \psi_2$, such that $\phi \blacktriangleleft \psi_1$, and $\gamma(p, \psi_2) \cap F \neq \emptyset$ with $\phi \blacktriangleleft \psi_1 \sqcup \{\!|p|\!\}$. By induction, we have that $|\phi| \leq |\psi_1| + 1$. Since A is 1-forking, we have that $|\psi_2| \geq 1$; hence, we conclude that $|\phi| \leq |\psi_1| + |\psi_2| = |\psi|$.

(ii) If $p \in Q_3$ and $\phi \in \mathbb{M}(Q_3)$ such that $\gamma_3(p, \phi) \neq \emptyset$, then by definition of γ_3 we have that $\phi = \psi^{\mathbf{s}}$ for some $\psi \in \mathbb{M}(Q)$. Hence, $\phi \in \mathbb{M}(Q^{\mathbf{s}})$.

(iii) Suppose that $p \in Q$ and $\phi \in \mathbb{M}(Q_3)$, and let $q \in \gamma_3(p^{\mathbf{s}}, \phi)$. By definition of γ_3, we have that $q \in \{r^{\mathbf{s}}, r^{\mathbf{p}}\}$ such that $r \in \gamma(p, \psi)$ for some $\psi \in \mathbb{M}(Q)$. Hence, $q \in Q^{\mathbf{s}} \cup Q^{\mathbf{p}}$. □

We are now set to prove that A_3 indeed satisfies the right properties.

Fact 8.C.16. *A_3 is 2-forking, parsimonious, and flat-branching.*

Proof. For 2-forking, suppose $p \in Q_3$ and $\phi \in \mathbb{M}(Q_3)$ such that $\gamma_3(p, \phi) \neq \emptyset$. Then by definition of γ_3 we find $\chi, \psi \in \mathbb{M}(Q)$ and $r \in Q$, such that $\phi = \chi^{\mathbf{s}}$, $\psi \blacktriangleleft \chi$, and $\gamma(r, \psi) \neq \emptyset$. Since A is 2-forking, we can conclude by Fact 8.C.15(i) that $2 \leq |\psi| \leq |\chi| = |\phi|$.

For parsimony, suppose $\gamma_3(p, \phi) \neq \emptyset$ and $q \in \phi$. Then by Fact 8.C.15(ii) we know that $q \in Q^{\mathbf{s}}$, so $q \notin F_3$. Since A_3 is 1-forking, $1 \notin L_{A_3}(q)$ by Fact 8.C.2, hence A_3 is parsimonious.

For flat-branching, suppose $p \in Q_3$ is a fork target. Then by Fact 8.C.15(ii), we know that $p \in Q^{\mathbf{s}}$. By Fact 8.C.15(iii), we can then conclude that $\gamma_3(p, \psi) \cap F_3 \subseteq (Q^{\mathbf{p}} \cup Q^{\mathbf{s}}) \cap F_3 = \emptyset$. □

We can now relate the runs of A_3 to those in A as follows.

Fact 8.C.17. *If $p \xrightarrow{U}_{A_3} q$, then the following hold:*

(i) If $p = p'^{\mathbf{s}}$ and $q \in \{q'^{\mathbf{s}}, q'^{\mathbf{p}}\}$, then $p' \xrightarrow{U}_A q'$.

(ii) If $p \in \{p'^{\mathbf{s}}, p'^{\mathbf{p}}\}$ and $q = \top$, then there exists a $q' \in F$ with $p' \xrightarrow{U}_A q'$.

Proof. We proceed by induction on $p \xrightarrow{U}_{A_3} q$. In the base, the case where $p \xrightarrow{U}_{A_3} q$ is trivial holds vacuously. Otherwise, if $p \xrightarrow{U}_{A_3} q$ because $U = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$ and $q \in \delta_3(p, \mathtt{a})$, then we know that $p \notin Q^{\mathbf{p}}$ by definition of δ_3. Therefore, assume $p = p'^{\mathbf{s}}$ for some $p' \in Q$. There are two cases.

- If $q \in \{q'^{\mathbf{s}}, q'^{\mathbf{p}}\}$, then it must be the case that $q' \in \delta(p', \mathtt{a})$, so $p' \xrightarrow{\mathtt{a}}_A q'$.
- If $q = \top$, then $\delta(p', \mathtt{a}) \cap F \neq \emptyset$. Choose $q' \in \delta(p', \mathtt{a}) \cap F$ to find that so $p' \xrightarrow{\mathtt{a}}_A q'$.

For the inductive step, there are again two cases.

- Suppose $p \xrightarrow{U}_{A_3} q$ because $U = V \cdot W$ and there exists an $r \in Q_3$ such that $p \xrightarrow{V}_{A_3} r$ and $r \xrightarrow{W}_{A_3} q$. Furthermore, we may assume w.l.o.g. that neither of these runs is trivial. Since $\top$ does not permit nontrivial runs, we have $r \in Q^{\mathbf{p}} \cup Q^{\mathbf{s}}$. Furthermore, $p \notin Q^{\mathbf{p}}$, because if $p \in Q^{\mathbf{p}}$ then $r = \top$; we set $p = p'^{\mathbf{s}}$. We do a case analysis on r.

  - If $r = r'^{\mathbf{p}}$, then necessarily $q = \top$. By the induction hypothesis we get that $p' \xrightarrow{V}_A r'$ and we find $q' \in F$ such that $r' \xrightarrow{W}_A q'$, hence $p' \xrightarrow{U}_A q'$.
  - If $r = r'^{\mathbf{s}}$, then by induction we get that $p' \xrightarrow{V \cdot W}_A r'$. We now look at q:
    * If $q \in \{q'^{\mathbf{s}}, q'^{\mathbf{p}}\}$, then by induction we have $r' \xrightarrow{W}_A q'$, so $p' \xrightarrow{V \cdot W}_A q'$.
    * If $q = \top$, then induction gives us $q' \in F$ such that $r' \xrightarrow{W}_A q'$, and thus $p' \xrightarrow{V \cdot W}_A q'$.

- Suppose $p \xrightarrow{U}_{A_3} q$ because $q_1, \dots, q_n \in Q_3$ with $q \in \gamma_3(p, \{\!|q_1, \dots, q_n|\!\})$, and $U = U_1 \parallel \cdots \parallel U_n$ such that for $1 \leq i \leq n$ we have $q_i \xrightarrow{U_i}_{A_3} \top$. Since A_3 is 1-forking and parsimonious, each U_i is non-empty by Fact 8.C.2. By definition of γ_3, for each $1 \leq i \leq n$ there exists a $q_i' \in Q$ such that $q_i = q_i'^{\mathbf{s}}$. By induction, we obtain for each $1 \leq i \leq n$ a $q_i'' \in F$ such that $q_i' \xrightarrow{U_i}_A q_i''$.

  On the one hand, suppose $p = p'^{\mathbf{s}}$ for $p' \in Q$. In that case, $q \in \{q'^{\mathbf{s}}, q'^{\mathbf{p}}\}$ for some $q' \in \gamma(p', \phi)$, with $\phi \blacktriangleleft \{\!|q_1, \dots, q_n|\!\}$. We show $p' \xrightarrow{U}_A q'$ by induction on $\blacktriangleleft$. In the base, $\phi = \{\!|q_1, \dots, q_n|\!\}$, and so the claim follows. In the inductive step, $\{\!|q_1, \dots, q_n|\!\} = \{\!|q_1, \dots, q_k|\!\} \sqcup \{\!|q_{k+1}, \dots, q_n|\!\}$ and there exists an $r \in Q$ such that $\gamma(r, \{\!|q_{k+1}, \dots, q_n|\!\}) \cap F \neq \emptyset$ and $\phi \blacktriangleleft \{\!|q_1, \dots, q_k|\!\} \sqcup \{\!|r|\!\}$. In that case, $r \xrightarrow{U_{k+1} \parallel \cdots \parallel U_n}_A r'$ for some $r' \in F$; hence, by induction, $p' \xrightarrow{U_1 \parallel \cdots \parallel U_n}_A q'$.

  On the other hand, if $p = p'^{\mathbf{p}}$ for some $p' \in Q$, then $q' = \top$ by definition of γ_3. Furthermore, there exists $q' \in \gamma(p', \phi) \cap F$ for some $\phi \in \mathbb{M}(Q)$ with $\phi \blacktriangleleft \{\!|q_1, \dots, q_n|\!\}$. A similar inductive argument to the previous case then shows that $p' \xrightarrow{U}_A q'$. □

Fact 8.C.18. *If $p \xrightarrow{U}_A q$ is nontrivial, then $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{s}}$ and $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{p}}$.*

Furthermore, if $q \in F$, then either $p^{\mathbf{s}} \xrightarrow{U}_{A_3} \top$ or $p^{\mathbf{p}} \xrightarrow{U}_{A_3} \top$.

Proof. We proceed by induction on $p \xrightarrow{U}_A q$. In the base, $p \xrightarrow{U}_A q$ because $U = \mathsf{a}$ for some $\mathsf{a} \in \Sigma$, and $q \in \delta(p, \mathsf{a})$. Thus $q^{\mathbf{s}}, q^{\mathbf{p}} \in \delta_3(p^{\mathbf{s}}, \mathsf{a})$, and hence $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{s}}$ and $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{p}}$. Furthermore, $q \in F$, then $\top \in \delta_3(p^{\mathbf{s}}, \mathsf{a})$, and hence $p^{\mathbf{s}} \xrightarrow{U}_{A_3} \top$. In the inductive step, there are two cases.

- If $p \xrightarrow{U}_A q$ because $U = V \cdot W$ and there exists an $r \in Q$ with $p \xrightarrow{V}_A r$ and $r \xrightarrow{W}_A q$, then we can assume without loss of generality that neither of these runs is trivial. By induction, we then find that $p^{\mathbf{s}} \xrightarrow{V}_{A_3} r^{\mathbf{s}}$ and $p^{\mathbf{s}} \xrightarrow{W}_{A_3} r^{\mathbf{s}}$, as well as $r^{\mathbf{s}} \xrightarrow{W}_{A_3} q^{\mathbf{s}}$ and $r^{\mathbf{s}} \xrightarrow{W}_{A_3} q^{\mathbf{p}}$. Putting this together, we have that $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{s}}$ and $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{p}}$.

Furthermore, if $q = \top$, then it suffices to prove that $r^{\mathbf{s}} \xrightarrow{W}_{A_3} \top$ or $r^{\mathbf{p}} \xrightarrow{W}_{A_3} \top$, which we obtain from $r \xrightarrow{W}_{A} q$ by induction.

- Suppose $p \xrightarrow{U}_{A} q$ because there exist $q_1, \dots, q_n \in Q$ such that $q \in \gamma(p, \{\!|q_1, \dots, q_n|\!\})$, and $U = U_1 \parallel \cdots \parallel U_n$ such that for $1 \leq i \leq n$ there exists a $q_i' \in F$ with $q_i \xrightarrow{U_i}_{A} q_i'$. Since A is parsimonious, we can assume without loss of generality that none of these runs is trivial.

  We then claim that, for $1 \leq i \leq n$, there exists a $\phi_i = \{\!|q_{i,1}, \dots, q_{i,n_i}|\!\} \in \mathbb{M}(Q)$ such that $\{\!|q_i|\!\} \blacktriangleleft \phi_i$, and $U_i = U_{i,1} \parallel \cdots \parallel U_{i,n_i}$, such that for $1 \leq i \leq n_i$ we have that ${q_{i,j}}^{\mathbf{s}} \xrightarrow{U_{i,j}}_{A_3} \top$. Applying the induction hypothesis to each $q_i \xrightarrow{U_i}_{A} q_i' \in F$, there are two cases to consider:

  - If ${q_i}^{\mathbf{s}} \xrightarrow{U_i}_{A_3} \top$, then we choose $n_i = 1$ and $q_{i,1} = q_i$ and $U_{i,1} = U_i$.
  - If ${q_i}^{\mathbf{p}} \xrightarrow{U_i}_{A_3} \top$, then by construction of A_3 this must be a parallel unit run. Consequently, there exist ${q_{i,1}}^{\mathbf{s}}, \dots, {q_{i,n_i}}^{\mathbf{s}} \in Q^{\mathbf{s}}$ with $\top \in \gamma_3({q_i}^{\mathbf{p}}, \{\!|q_{i,1}, \dots, q_{i,n_i}|\!\})$, and $U_i = U_{i,1} \parallel \cdots \parallel U_{i,n_i}$ such that for $1 \leq j \leq n_i$ we have that ${q_{i,j}}^{\mathbf{s}} \xrightarrow{U_{i,j}}_{A_3} \top$. By definition of γ_3, we then obtain $\psi \in \mathbb{M}(Q)$ such that $\gamma(q_i, \psi) \cap F \neq \emptyset$ and $\psi \blacktriangleleft \{\!|q_{i,1}, \dots, q_{i,n_i}|\!\}$. A straightforward inductive argument on the definition of $\blacktriangleleft$ shows that it is transitive; hence, since $\{\!|q_i|\!\} \blacktriangleleft \psi$, we have that $\{\!|q_i|\!\} \blacktriangleleft \{\!|q_{i,1}, \dots, q_{i,n_i}|\!\}$.

  Using the above, it follows that $\{\!|q_1, \dots, q_n|\!\} \blacktriangleleft \{\!|q_{1,1}, \dots, q_{n,n_n}|\!\}$. Hence,

$$q^{\mathbf{s}}, q^{\mathbf{p}} \in \gamma_3(p^{\mathbf{s}}, \{\!|{q_{1,1}}^{\mathbf{s}}, \dots, {q_{n,n_n}}^{\mathbf{s}}|\!\})$$

  Since $U = U_{1,1} \parallel \cdots \parallel U_{n_n}$, it follows that $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{s}}$ and $p^{\mathbf{s}} \xrightarrow{U}_{A_3} q^{\mathbf{p}}$.

  Furthermore, if $q \in F$, then $\top \in \gamma_3(p^{\mathbf{p}}, \{\!|{q_{1,1}}^{\mathbf{s}}, \dots, {q_{n,n_n}}^{\mathbf{s}}|\!\})$, and hence $p^{\mathbf{p}} \xrightarrow{U}_{A_3} \top$. □

We are now ready to show that our construction preserves languages. More specifically, Facts 8.C.17 and 8.C.18 together imply that for $p \in Q$, we have

$$L_{A_1}(q) = L_{A_3}(q^{\mathbf{p}}) \cup L_{A_3}(q^{\mathbf{s}}) \cup \begin{cases} L_{A_3}(\top) & q \in F \\ \emptyset & \text{otherwise} \end{cases}$$

To see that our construction preserves fork-acyclicity and boundedness, one can show that if $p, q \in Q$ are such that $p^{\mathbf{s}} \preceq_{A_3} q^{\mathbf{s}}$, $p^{\mathbf{s}} \preceq_{A_3} q^{\mathbf{p}}$, $p^{\mathbf{p}} \preceq_{A_3} q^{\mathbf{s}}$ or $p^{\mathbf{p}} \preceq_{A_3} q^{\mathbf{p}}$, then $p \preceq_{A_3} q$. A fork cycle in A_3 thus gives rise to a fork cycle in A, which means that if A is fork-acyclic, then so is A_3. Furthermore, the support of a state $q^{\mathbf{s}}$ or $q^{\mathbf{p}}$ in A_3 is contained in $\{p^{\mathbf{s}}, p^{\mathbf{p}} : p \in \pi_A(q)\} \cup \{\top\}$; since the latter is finite when A is bounded, it follows that A_3 must also be bounded. □

Chapter 9

Parallel Star

The iteration operator of series-rational expressions, denoted $(-)^*$, takes a program e and turns it into the program e^*, which runs e some number of times. This is useful for describing loops; for instance, we have seen that it can be used to implement conditional loops of the form $(b \cdot e)^* \cdot \overline{b}$, where b is the assertion that guards the loop. As we alluded to several times before, there is a parallel analogue to this operator, denoted $(-)^\dagger$ (referred to as *parallel star* or sometimes *dagger*). The intuition to this operator is that it takes a program e and turns it into the program $e^\dagger$, which runs some number of parallel copies of e. While such a pattern may be rare in typical program constructs, it can still occur; for instance, think of a web server that has to run some number of parallel threads to serve requests, or a distributed program that starts some number of agents to complete a certain task. In this chapter, we investigate the consequences of adding the parallel star operator to series-rational expressions with regard to the Kleene theorem that we saw in Chapter 7. In a nutshell, it turns out that this theorem can be extended to include a parallel star, provided that we also relax the restriction of fork-acyclicity. Before we get into the details of this construction, let us start by formally defining this extended syntax as well as its semantics.

Definition 9.1 (Syntax)**.** The set of *series-parallel rational expressions*, or *spr-expressions* for short, is denoted by $\mathcal{T}_{\mathsf{SPR}}$ and generated by the grammar

$$e, f ::= 0 \mid 1 \mid \mathtt{a} \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^* \mid e^\dagger$$

We can extend the semantics $[\![-]\!] : \mathcal{T} \to 2^{\mathsf{SP}}$ to $[\![-]\!]_{\mathsf{SPR}} : \mathcal{T}_{\mathsf{SPR}} \to 2^{\mathsf{SP}}$ as follows:

$$\begin{array}{llll}
[\![0]\!]_{\mathsf{SPR}} = \emptyset & [\![\mathtt{a}]\!]_{\mathsf{SPR}} = \mathtt{a} & [\![e \cdot f]\!]_{\mathsf{SPR}} = [\![e]\!]_{\mathsf{SPR}} \cdot [\![f]\!]_{\mathsf{SPR}} & [\![e^*]\!]_{\mathsf{SPR}} = [\![e]\!]_{\mathsf{SPR}}^* \\
[\![1]\!]_{\mathsf{SPR}} = \{1\} & [\![e + f]\!]_{\mathsf{SPR}} = [\![e]\!]_{\mathsf{SPR}} + [\![f]\!]_{\mathsf{SPR}} & [\![e \parallel f]\!]_{\mathsf{SPR}} = [\![e]\!]_{\mathsf{SPR}} \parallel [\![f]\!]_{\mathsf{SPR}} & [\![e^\dagger]\!]_{\mathsf{SPR}} = [\![e]\!]_{\mathsf{SPR}}^\dagger
\end{array}$$

where the parallel star of a pomset language $L \subseteq \mathsf{Pom}$ is given by

$$L^\dagger = \bigcup_{n \in \mathbb{N}} L^{(n)} \quad \text{where} \quad L^{(0)} = \{1\} \quad \text{and} \quad L^{(n+1)} = L^{(n)} \parallel L$$

It should be clear that when $e \in \mathcal{T}$, also $e \in \mathcal{T}_{\mathsf{SPR}}$, and furthermore $[\![e]\!] = [\![e]\!]_{\mathsf{SPR}}$. Thus, if $e, f \in \mathcal{T}_{\mathsf{SPR}}$ do not include any occurrences of $(-)^\dagger$, we can soundly (and completely) reason about them using $\equiv$. The question then arises: how do we reason about equivalence between spr-expressions where the new operator does occur? The intuitive thing to do is to add new axioms that are analogous to the axioms for the Kleene star operator. More concretely, we have the following.

Definition 9.2 (Extended bi-Kleene algebra)**.** An *extended bi-Kleene algebra congruence*, or *EBKA congruence* for short, is a BKA congruence $\approx$ on $\mathcal{T}_{\mathsf{SPR}}$ w.r.t. all operators, such that for all $e, f, g \in \mathcal{T}$:

$$1 + e \parallel e^\dagger \approx e^\dagger \qquad e + f \parallel g \lessapprox g \implies f^\dagger \parallel e \lessapprox g \qquad (\text{where } e \lessapprox f \iff e + f \approx f)$$

We write $\equiv_{\mathsf{SPR}}$ for the smallest EBKA congruence, and $e \leqq_{\mathsf{SPR}} f$ whenever $e + f \equiv_{\mathsf{SPR}} f$.

Analogous to previous notation, we define the relation $\doteq_{\mathsf{SPR}}$ on $\mathcal{T}_{\mathsf{SPR}}(2^{\mathsf{SP}})$ as relating expressions over pomset languages involving the operators of spr-expressions that have the same interpretation. For instance, we have for $L, K \subseteq \mathsf{SP}$ that $L \parallel K \doteq_{\mathsf{SPR}} K \parallel L$, and it is also not hard to show that $1 + L \parallel L^\dagger \doteq_{\mathsf{SPR}} L^\dagger$. Laurence and Struth [LS14] showed that their completeness result for series-rational expressions (Theorem 3.51) can be stated more generally as follows.

Theorem 9.3 [LS14]**.** *The relation $\doteq_{\mathsf{SPR}}$ is an EBKA congruence on $\mathcal{T}_{\mathsf{SPR}}(2^{\mathsf{SP}})$; in particular, if $e, f \in \mathcal{T}_{\mathsf{SPR}}$ such that $e \equiv_{\mathsf{SPR}} f$, then $[\![e]\!]_{\mathsf{SPR}} = [\![f]\!]_{\mathsf{SPR}}$. Conversely, if $[\![e]\!]_{\mathsf{SPR}} = [\![f]\!]_{\mathsf{SPR}}$, then $e \equiv_{\mathsf{SPR}} f$.*

Remark 9.4. In [LS14] it is also shown that equivalence of spr-expressions is decidable.

Finally, the results about series-rational systems and their solutions can similarly be extended to series-parallel expressions quite easily — the proof of Theorem 3.60 can be copied almost verbatim.

Definition 9.5 (Series-parallel rational systems)**.** Let Q be a finite set. A *series-parallel rational system*, or *spr-system* for short, on Q is a tuple $\mathcal{S} = \langle M, b \rangle$, where $M : Q^2 \to \mathcal{T}_{\mathsf{SPR}}$ and $b : Q \to \mathcal{T}_{\mathsf{SPR}}$. Let $\approx$ be a EBKA congruence on $\mathcal{T}_{\mathsf{SPR}}(\Delta)$ with $\Sigma \subseteq \Delta$. We call $s : Q \to \mathcal{T}_{\mathsf{SPR}}(\Delta)$ a *$\approx$-solution* to $\mathcal{S}$ if for every $q \in Q$, it holds that:

$$b(q) + \sum_{q' \in Q} M(q, q') \cdot s(q') \lessapprox s(q)$$

Lastly, s is the *least* $\approx$-solution if, for every $\approx$-solution s' to $\mathcal{S}$ and every $q \in Q$ we have $s(q) \lessapprox s'(q)$.

Theorem 9.6. *Let $\mathcal{S} = \langle M, b \rangle$ be an spr-system on Q. We can construct an $s : Q \to \mathcal{T}_{\mathsf{SPR}}$ that is the least $\approx$-solution to $\mathcal{S}$ for any EBKA congruence $\approx$ on $\mathcal{T}_{\mathsf{SPR}}(\Delta)$ with $\Sigma \subseteq \Delta$.*

9.1 Well-nested pomset automata

Before we embark on our quest to establish a Kleene theorem for series-parallel rational expressions w.r.t. pomset automata, let us take a moment to establish the boundary conditions of this correspondence. For one thing, fork-acyclic and finite pomset automata are not expressive enough. To see why this is the case, note that if any series-parallel rational expression could be implemented by a fork-acyclic and finite pomset automaton, then our Kleene theorem for series-rational expressions (specifically, Theorem 7.41) implies that it could be implemented by a series-rational expression. The latter cannot be true; intuitively, this is because the parallel star allows an unbounded number of events to occur in parallel. To formalise this, we use the notion of *width* [LW00].

Definition 9.7 (Pomset width)**.** The *width* of a finite pomset $U = [\mathbf{u}] \in \mathsf{Pom}$ is the size of the largest $\leq_{\mathbf{u}}$-antichain, i.e., the maximal $n \in \mathbb{N}$ s.t. there exist $u_1, u_2, \dots, u_n \in S_{\mathbf{u}}$ unrelated by $\leq_{\mathbf{u}}$.

Example 9.8. Let $U = \mathtt{a} \parallel \mathtt{b} \cdot \mathtt{c}$; then U has width 2, because the nodes labelled by $\mathtt{a}$ and $\mathtt{b}$ are unrelated, and the third node (labelled by $\mathtt{c}$) is related to the node labelled by $\mathtt{b}$.

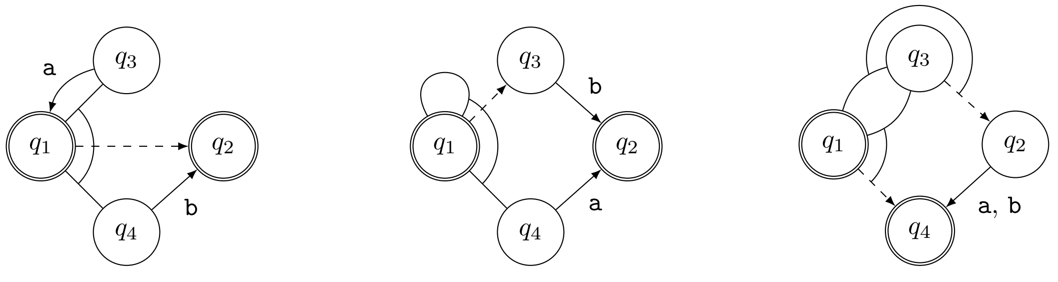
One can show that for every series-rational language L there exists an $n \in \mathbb{N}$ such that for $U \in L$, the width of U is bounded from above by n [LW00, Lemma 1.7]. On the other hand, there exist series-parallel rational languages — such as $\mathtt{a}^\dagger$ — that do not have this property; after all, for every $n \in \mathbb{N}$ we can simply take the n-fold parallel composition of $\mathtt{a}$ to find a pomset of width $n + 1$ in $\mathtt{a}^\dagger$, and hence such an upper bound cannot exist. From this, it follows that series-parallel rational languages are strictly more expressive than series-rational languages, and we will need something more expressive than fork-acyclic and finite pomset automata in our Kleene theorem.

On the other hand, general pomset automata are too powerful to correspond to spr-expressions; after all, the languages of words expressed by spr-expressions are simply rational languages, whereas pomset automata can be used to express non-rational languages, as we saw in Chapter 8.

Orthogonally, some pomset languages cannot be expressed by series-parallel rational expressions for structural reasons. To explain this, we need the notion of *depth*; intuitively, the depth of a pomset is a measure for the nesting between parallel and sequential composition.

Definition 9.9 (Pomset depth) [LS14]**.** The *depth* of $U \in \mathsf{SP}$, denoted $d(U)$, is defined inductively:

- If U is empty or primitive, then $d(U) = 0$.
- If U is sequential with sequential factorisation $U_1, \dots, U_n$, then $d(U) = 1 + \max_{1 \leq i \leq n} d(U_i)$.
- If U is parallel with parallel factorisation $\{\!|U_1, \dots, U_n|\!\}$, then $d(U) = 1 + \max_{1 \leq i \leq n} d(U_i)$.



(a) A_0. (b) A_1. (c) A_2.

Figure 9.1: Pomset automata accepting languages of unbounded depth.

Example 9.10. The pomset $\mathtt{a} \cdot \mathtt{b}$ has depth one, while $\mathtt{a} \cdot (\mathtt{b} \parallel \mathtt{c})$ has depth two, as does $\mathtt{a} \parallel (\mathtt{b} \cdot \mathtt{c})$.

One can show that for every series-parallel rational language L there exists an $n \in \mathbb{N}$ such that for $U \in L$, the width of U is bounded from above by n [LS14, Lemma 13]. On the other hand, pomset automata can be used to express a litany of languages where there is no upper bound on the depth of accepted pomsets; for instance, consider the automata in Figure 9.1. As we argued in Example 7.9, the pomset automaton A_0 in Figure 9.1a accepts the pomsets $\{U_n\}_{n \in \mathbb{N}}$ where

$$U_n = \begin{cases} 1 & n = 0 \\ (\mathtt{a} \cdot U_{n-1}) \parallel \mathtt{b} & n > 0 \end{cases}$$

Similarly, we can show that A_1 (Figure 9.1b) and A_2 (Figure 9.1c) respectively accept the families of pomsets $\{V_n\}_{n \in \mathbb{N}}$ and $\{W_n\}_{n \in \mathbb{N}}$, defined inductively by

$$V_n = \begin{cases} 1 & n = 0 \\ (V_{n-1} \parallel \mathtt{a}) \cdot \mathtt{b} & n > 0 \end{cases} \qquad W_n = \begin{cases} 1 & n = 0 \\ (W_{n-1} \cdot \mathtt{a}) \parallel \mathtt{b} & n > 0 \end{cases}$$

It is not hard to show that, for $n \in \mathbb{N}$, we have that $d(U_n) = d(V_n) = d(W_n) = 2n$, which shows that the depth of these pomset languages cannot be bounded from above.

We thus have to walk a tightrope: on the one hand, we want PAs to express languages of unbounded *width*, but not languages of unbounded *depth*. The PAs from Figure 9.1 already give us some idea of the patterns that we may want to exclude, which we semi-formally describe as follows:

- forks into states that transition back to their origin (as is the case in Figure 9.1a),
- destinations of fork transitions having their own transitions (as is the case in Figure 9.1b), or
- mutual forks (as seen in Figure 9.1c as well as the PAs that model CFGs in Chapter 8).

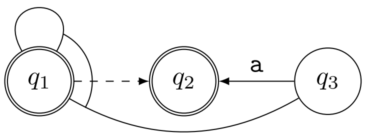


Figure 9.2: A pomset automaton implementing the language $\mathtt{a}^\dagger$.

For a positive example, i.e., a PA that implements a language of unbounded width but bounded depth, consider the PA in Figure 9.2. Here, we see a fork cycle in state q_1 (but this cycle is as short as can be), the destination of the fork (q_2) does not have any further transitions, and there are no mutual forks between states. Thus, the PA from Figure 9.2 does not run afoul of the antipatterns for unbounded depth from Figure 9.1. We can formalise this distinction as follows.

Definition 9.11 (Recursive states)**.** Let $A = \langle Q, F, \delta, \gamma \rangle$ be a PA. We call $q \in Q$ *recursive* if:

(i) for all $\mathtt{a} \in \Sigma$ and $\phi \in \mathbb{M}(Q)$, if $q' \in \delta(q, \mathtt{a})$ or $q' \in \gamma(q, \phi)$, then $q' \prec_A q$, and

(ii) if $\phi \in \mathbb{M}(Q)$ with $q' \in \gamma(q, \phi)$, then either (a) $\phi = \{\!|q|\!\} \sqcup \psi$, such that for all $r \in \psi$ we have $r \prec_A q$, and q' does not have any outgoing transitions, or (b) for all $r \in \phi$ we have $r \prec_A q$.

In other words, recursive states are states that transition only into strictly supporting states (as in req. (i)), and where forks either have a tight loop and continue in states without outgoing transitions (case (a) of req. (ii)) or where all fork targets are strictly supporting (case (b) in req. (ii)).

Example 9.12. In Figures 9.1a and 9.1c, q_1 is not recursive because the parallel transition from q_1 violates the last requirement. In Figure 9.1b, q_1 is not recursive, because q_3 allows a further (sequential) transition to q_2. On the other hand, q_1 in Figure 9.2 is recursive.

Of course, having recursive states exclusively would mean that there cannot be any cycle in the support of states, which would severely limit expressiveness. Thus, we also need to allow states that do not implement a parallel star of a language; for this, we mimic fork-acyclicity. Pomset automata where states can be labelled as either of these are the ones we are aiming for.

Definition 9.13 (Progressive states and well-nested PAs)**.** Let $A = \langle Q, F, \delta, \gamma \rangle$ be a PA. We call $q \in Q$ *progressive* if, whenever $\phi \in \mathbb{M}(Q)$ is such that $\gamma(q, \phi) \neq \emptyset$, we have for all $r \in \phi$ that $r \prec_A q$. We say that A is *well-nested* if every state is either progressive or recursive.

Example 9.14. The states q_2 and q_3 in Figure 9.2 are progressive. Since q_1 is recursive, this pomset automaton is well-nested. On the other hand, since the state q_1 in all of the pomset automata from Figure 9.1 is not progressive (on account of the fork), none of these PAs are well-nested.

9.2 An extended Kleene theorem

In the previous section, we proposed well-nested and finite pomset automata as a putative operational model for series-parallel rational expressions. In this section, we show that the languages recognised by spr-expressions are precisely those described by well-nested and finite pomset automata. We do this by means of a Kleene theorem, as in Chapter 7, which is similarly split between a translation from expressions to automata, and automata to expressions.

In comparison with other works, the result presented here is the first two-way correspondence between spr-expressions and an operational formalism. In particular, Lodaya and Weil [LW00] provided a translation from spr-expressions to branching automata, but not the other way around, while Jipsen and Moshier [JM16] do not include the parallel star at all. In comparison with [LW00], the same differences as in Chapter 7 apply: our conversion from spr-expressions to pomset automata is based on Antimirov's partial derivatives rather than Thompson's inductive approach, and may therefore be more suitable for lazy exploration of the state space. Lastly, 1-safe Petri nets as used in [Gra81; LRR03; BPS17] are not suitable for capturing pomset languages of unbounded width; conceptually, this is because these Petri nets have at most one token per place, and recognising a pomset of unbounded width means that there cannot be an upper bound on the number of tokens active — which would mean that the Petri net would need an unbounded number of places.

9.2.1 Expressions to automata

To translate series-parallel rational expressions to well-nested and finite pomset automata, we take the same approach as in Section 7.2: we define a transition structure on the space of all series-parallel rational expressions, show that the resulting automaton is well-nested and bounded, which means that we can restrict any one state/expression to a well-nested and finite PA.

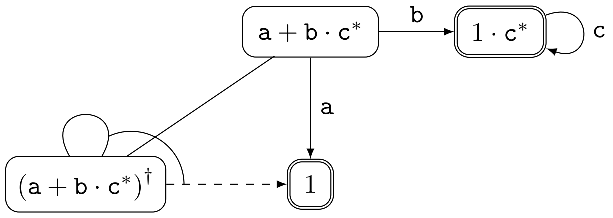
For this strategy to work, we must first ascertain that the restriction of a well-nested pomset automaton to a support-closed set preserves well-nestedness. This turns out to be the case.

Lemma 9.15. *Let A be a PA, with Q' be support-closed in A. If A is well-nested, then so is $A[Q']$.*

In Section 7.2, we chose $\mathcal{F}$ as the set of accepting states, because these were the expressions whose semantics contains the empty pomset. We can extend this to spr-expressions as follows.

Definition 9.16. We define $\mathcal{F}_{\mathsf{SPR}}$ as the smallest subset of $\mathcal{T}_{\mathsf{SPR}}$ satisfying the following rules

$$\frac{}{1 \in \mathcal{F}_{\mathsf{SPR}}} \qquad \frac{e \in \mathcal{T}_{\mathsf{SPR}} \qquad f \in \mathcal{F}_{\mathsf{SPR}}}{e + f, f + e \in \mathcal{F}_{\mathsf{SPR}}} \qquad \frac{e, f \in \mathcal{F}_{\mathsf{SPR}}}{e \cdot f, e \parallel f \in \mathcal{F}_{\mathsf{SPR}}} \qquad \frac{e \in \mathcal{T}_{\mathsf{SPR}}}{e^*, e^\dagger \in \mathcal{F}_{\mathsf{SPR}}}$$



$\mathtt{a} + \mathtt{b} \cdot \mathtt{c}^*$
b
$1 \cdot \mathtt{c}^*$
c
a
$(\mathtt{a} + \mathtt{b} \cdot \mathtt{c}^*)^\dagger$
1

Figure 9.3: Part of the series-parallel rational syntactic pomset automaton.

Convention 9.17. We extend our earlier notational conventions about constructing sets of series-rational expressions to series-parallel rational expressions. Concretely, when $S \subseteq \mathcal{T}_{\mathsf{SPR}}$ and $e \in \mathcal{T}_{\mathsf{SPR}}$, we use $e \star S$ to denote S when $e \in \mathcal{F}_{\mathsf{SPR}}$ and $\emptyset$ otherwise, and we use $S \fatsemi e$ to denote $\{e' \cdot e \ : \ e' \in S\}$.

We can now define a transition structure on series-parallel rational expressions. The intuition is analogous to the previous construction: every state $e \in \mathcal{T}_{\mathsf{SPR}}$ is meant to accept the language $\llbracket e \rrbracket_{\mathsf{SPR}}$, and transitioning from e to e' reading U means that every pomset $V \in \llbracket e' \rrbracket_{\mathsf{SPR}}$ can be used to complete U into a pomset accepted by e. In particular, since every $e \in \mathcal{F}_{\mathsf{SPR}}$ can trivially transition to itself, and $\llbracket e \rrbracket$ for $e \in \mathcal{F}_{\mathsf{SPR}}$ contains the empty pomset, we choose $\mathcal{F}_{\mathsf{SPR}}$ as accepting states.

Definition 9.18 (Derivatives)**.** We define $\delta_{\mathsf{SPR}} : \mathcal{T}_{\mathsf{SPR}} \times \Sigma \to 2^{\mathcal{T}_{\mathsf{SPR}}}$ inductively, as follows

$$\begin{aligned}
\delta_{\mathsf{SPR}}(0, \mathtt{a}) &= \emptyset & \delta_{\mathsf{SPR}}(e \cdot f, \mathtt{a}) &= \delta_{\mathsf{SPR}}(e, \mathtt{a}) \fatsemi f \ \cup \ e \star \delta_{\mathsf{SPR}}(f, \mathtt{a}) \\
\delta_{\mathsf{SPR}}(1, \mathtt{a}) &= \emptyset & \delta_{\mathsf{SPR}}(e \parallel f, \mathtt{a}) &= \emptyset \\
\delta_{\mathsf{SPR}}(\mathtt{b}, \mathtt{a}) &= \{1 \ : \ \mathtt{a} = \mathtt{b}\} & \delta_{\mathsf{SPR}}(e^*, \mathtt{a}) &= \delta_{\mathsf{SPR}}(e, \mathtt{a}) \fatsemi e^* \\
\delta_{\mathsf{SPR}}(e + f, \mathtt{a}) &= \delta_{\mathsf{SPR}}(e, \mathtt{a}) \cup \delta_{\mathsf{SPR}}(f, \mathtt{a}) & \delta_{\mathsf{SPR}}(e^\dagger, \mathtt{a}) &= \emptyset
\end{aligned}$$

We also define $\gamma_{\mathsf{SPR}} : \mathcal{T}_{\mathsf{SPR}} \times \mathbb{M}(\mathcal{T}_{\mathsf{SPR}}) \to 2^{\mathcal{T}_{\mathsf{SPR}}}$ inductively, as follows:

$$\begin{aligned}
\gamma_{\mathsf{SPR}}(0, \phi) &= \emptyset & \gamma_{\mathsf{SPR}}(e \cdot f) &= \gamma_{\mathsf{SPR}}(e, \phi) \fatsemi f \ \cup \ e \star \gamma_{\mathsf{SPR}}(f, \phi) \\
\gamma_{\mathsf{SPR}}(1, \phi) &= \emptyset & \gamma_{\mathsf{SPR}}(e \parallel f, \phi) &= \{1 \ : \ \phi = \{\!|e, f|\!\}\} \\
\gamma_{\mathsf{SPR}}(\mathtt{b}, \phi) &= \emptyset & \gamma_{\mathsf{SPR}}(e^*, \phi) &= \gamma_{\mathsf{SPR}}(e, \phi) \fatsemi e^* \\
\gamma_{\mathsf{SPR}}(e + f, \phi) &= \gamma_{\mathsf{SPR}}(e, \phi) \cup \gamma_{\mathsf{SPR}}(f, \phi) & \gamma_{\mathsf{SPR}}(e^\dagger, \phi) &= \left\{1 \ : \ \phi = \{\!|e, e^\dagger|\!\}\right\}
\end{aligned}$$

The *(series-parallel rational) syntactic PA* is then given by $A_{\mathsf{SPR}} = \langle \mathcal{T}_{\mathsf{SPR}}, \mathcal{F}_{\mathsf{SPR}}, \delta_{\mathsf{SPR}}, \gamma_{\mathsf{SPR}} \rangle$.

Example 9.19. We have drawn part of the series-parallel rational syntactic PA, specifically, the support of $e = (\mathtt{a} + \mathtt{b} \cdot \mathtt{c}^*)^\dagger$, in Figure 9.3. There, we see that $1 \in \gamma_{\mathsf{SPR}}(e, \{\!|\mathtt{a} + \mathtt{b} \cdot \mathtt{c}^*, e|\!\})$. The other states are sr-expressions, and their transitions match the series-rational syntactic PA.

It remains to show that the syntactic PA faithfully implements the language of each spr-expression, and that it is well-nested and bounded. For the former, we start by noting that, because the transition functions from the series-parallel rational syntactic PA are set up similarly to the series-rational syntactic PA, the proofs of Lemmas 7.25 to 7.28 go through for series-parallel rational expressions. It remains to prove a similar statement for expressions of the form $e^\dagger$, as follows.

Lemma 9.20. *Let $e \in \mathcal{T}_{\mathsf{SPR}}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e^\dagger \xrightarrow{U}_{\mathsf{SPR}} f$.

(ii) $U = U_1 \parallel \cdots \parallel U_n$, such that for $1 \leq i \leq n$ there exists $f_i \in \mathcal{F}_{\mathsf{SPR}}$ with $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$.

With these in hand, we can then prove a statement analogous to Lemma 7.29, as follows.

Lemma 9.21. *Let $e, f \in \mathcal{T}_{\mathsf{SPR}}$. The following hold:*

$$L_{\mathsf{SPR}}(e + f) = L_{\mathsf{SPR}}(e) + L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e \cdot f) = L_{\mathsf{SPR}}(e) \cdot L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e^*) = L_{\mathsf{SPR}}(e)^*$$
$$L_{\mathsf{SPR}}(e \parallel f) = L_{\mathsf{SPR}}(e) \parallel L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e^\dagger) = L_{\mathsf{SPR}}(e)^\dagger$$

A straightforward inductive proof then leads to the desired statement of language correctness.

Lemma 9.22. *For all $e \in \mathcal{T}_{\mathsf{SPR}}$, we have $L_{\mathsf{SPR}}(e) = [\![e]\!]_{\mathsf{SPR}}$.*

To show that the syntactic PA is bounded, we can employ the same tactic as before, that is, we find a finite and support-closed set for every state e; since this overestimates the support of e, $\pi_{\mathsf{SPR}}(e)$ must be finite. The overestimation we choose is a simple extension of the one used before.

Definition 9.23. We define $R : \mathcal{T}_{\mathsf{SPR}} \to 2^{\mathcal{T}_{\mathsf{SPR}}}$ inductively, as follows

$$\begin{array}{lll} R(0) = \{0\} & R(e_1 + e_2) = R(e_1) \cup R(e_2) \cup \{e\} & R(e_1^*) = R(e_1) \fatsemi e_1^* \cup R(e_1) \cup \{e_1^*\} \\ R(1) = \{1\} & R(e_1 \cdot e_2) = R(e_1) \fatsemi e_2 \cup R(e_1) \cup R(e_2) & R(e_1^\dagger) = R(e_1) \cup \{e_1^\dagger, 1\} \\ R(\mathtt{a}) = \{\mathtt{a}, 1\} & R(e_1 \parallel e_2) = R(e_1) \cup R(e_2) \cup \{e_1 \parallel e_2, 1\} & \end{array}$$

It should be clear that for $e \in \mathcal{T}_{\mathsf{SPR}}$, $R(e)$ is finite. It is also support-closed; since the definition of R extends Definition 7.33, the proof of the following is a simple extension of Lemma 7.34.

Lemma 9.24. *For every $e \in \mathcal{T}_{\mathsf{SPR}}$, we have that $e \in R(e)$ and $R(e)$ is support-closed.*

Consequently, the series-parallel rational syntactic PA is bounded.

To argue well-nestedness, we need a proxy to argue that a state *does not* support another state, like the parallel depth $\mathsf{d}_{\parallel}(-)$. In this case, we have two such proxies; one is an extension of the $\dagger$-depth to spr-expressions, and another is its analogue for the parallel star.

Definition 9.25 (†-depth)**.** We extend the domain of $\mathsf{d}_{\|}(-)$ to $\mathcal{T}_{\mathsf{SPR}}$ by setting $\mathsf{d}_{\|}(e^{\dagger}) = \mathsf{d}_{\|}(e)$. Furthermore, we define the †-*depth* $\mathsf{d}_{\dagger}(-) : \mathcal{T}_{\mathsf{SPR}} \to \mathbb{N}$ inductively, as follows:

$$\begin{aligned} \mathsf{d}_{\dagger}(0) &= 0 & \mathsf{d}_{\dagger}(e_0 \cdot e_1) &= \max(\mathsf{d}_{\dagger}(e_0), \mathsf{d}_{\dagger}(e_1)) \\ \mathsf{d}_{\dagger}(1) &= 0 & \mathsf{d}_{\dagger}(e_0 \parallel e_1) &= \max(\mathsf{d}_{\dagger}(e_0), \mathsf{d}_{\dagger}(e_1)) \\ \mathsf{d}_{\dagger}(\mathsf{a}) &= 0 & \mathsf{d}_{\dagger}(e_0 + e_1) &= \max(\mathsf{d}_{\dagger}(e_0), \mathsf{d}_{\dagger}(e_1)) \\ \mathsf{d}_{\dagger}(e_0^*) &= \mathsf{d}_{\dagger}(e_0) & \mathsf{d}_{\dagger}(e_0^{\dagger}) &= \mathsf{d}_{\dagger}(e_0) + 1 \end{aligned}$$

Both depth functions can be related to support by noting that the depth of a state is at most the depth of a state it supports. Formally, we have the following.

Lemma 9.26. *For $e, f \in \mathcal{T}_{\mathsf{SPR}}$ such that $e \preceq_{\mathsf{SPR}} f$, we have $\mathsf{d}_{\|}(e) \leq \mathsf{d}_{\|}(f)$ and $\mathsf{d}_{\dagger}(e) \leq \mathsf{d}_{\dagger}(f)$.*

Because of how the derivatives for spr-expressions of the form $e^{\dagger}$ are set up, mutual support involving such an spr-expression must be trivial. More precisely, we can prove the following.

Lemma 9.27. *Let $e, f \in \mathcal{T}_{\mathsf{SPR}}$. If $e \preceq_{\mathsf{SPR}} f^{\dagger}$ and $\mathsf{d}_{\dagger}(e) = \mathsf{d}_{\dagger}(f^{\dagger})$, then $e = f^{\dagger}$.*

Proof. We proceed by induction on $\preceq_{\mathsf{SPR}}$. In the base, there are three cases to consider:

- If $e \in \delta_{\mathsf{SPR}}(f^{\dagger}, \mathsf{a})$ for some $\mathsf{a} \in \Sigma$ or $e \in \gamma_{\mathsf{SPR}}(f^{\dagger}, \phi)$ for some $\phi \in \mathbb{M}(\mathcal{T}_{\mathsf{SPR}})$, then by definition of our derivatives we know that $e = 1$. In that case, $\mathsf{d}_{\dagger}(e) < 1 + \mathsf{d}_{\dagger}(f) = \mathsf{d}_{\dagger}(f)$, which contradicts the premise that $\mathsf{d}_{\dagger}(e) = \mathsf{d}_{\dagger}(f)$; we can therefore disregard this case.
- If $e \in \phi$ such that $\gamma_{\mathsf{SPR}}(f^{\dagger}, \phi) \neq \emptyset$, there are two subcases. On the one hand, if $e = f^{\dagger}$, then the claim holds immediately. On the other hand, if $e = f$, then $\mathsf{d}_{\dagger}(e) = \mathsf{d}_{\dagger}(f) < \mathsf{d}_{\dagger}(f^{\dagger})$, which contradicts the premise that $\mathsf{d}_{\dagger}(e) = \mathsf{d}_{\dagger}(f^{\dagger})$; we can therefore disregard this case as well.

For the inductive step, assume that $e \preceq_{\mathsf{SPR}} f^{\dagger}$ because $e \preceq_{\mathsf{SPR}} g \preceq_{\mathsf{SR}} f^{\dagger}$ for some $g \in \mathcal{T}_{\mathsf{SPR}}$. By Lemma 9.26, we know that $\mathsf{d}_{\dagger}(e) \leq \mathsf{d}_{\dagger}(g) \leq \mathsf{d}_{\dagger}(f^{\dagger}) = \mathsf{d}_{\dagger}(e)$, and hence $\mathsf{d}_{\dagger}(g) = \mathsf{d}_{\dagger}(f^{\dagger}) = \mathsf{d}_{\dagger}(e)$. By induction, we then find that $g = f^{\dagger}$; applying induction to $e \preceq_{\mathsf{SPR}} g$, we find that $e = g = f^{\dagger}$ as well.

Since the main claim implies that $e \preceq_{\mathsf{SPR}} f^{\dagger}$ and $\mathsf{d}_{\dagger}(e) = \mathsf{d}_{\dagger}(f^{\dagger})$ by Lemma 9.26, we are done. □

Using Lemma 9.26, we can also show that the structure of the series-parallel rational syntactic PA is such that every fork has its targets either strictly supporting the origin, or exactly one of the fork targets supports the origin, and is of the form $e^{\dagger}$. Formally, we have the following.

Lemma 9.28. *Let $e \in \mathcal{T}_{\mathsf{SPR}}$ and $\phi \in \mathbb{M}(\mathcal{T}_{\mathsf{SPR}})$ such that $\gamma_{\mathsf{SPR}}(e, \phi) \neq \emptyset$. Then $\phi = \{\!\!\{f, g\}\!\!\}$ such that either (i) $f \prec_{\mathsf{SPR}} e$ as well as $g \prec_{\mathsf{SPR}} e$, or (ii) $f \prec_{\mathsf{SPR}} e$ and $g = h^{\dagger}$ for some $h \in \mathcal{T}_{\mathsf{SPR}}$.*

With this property in hand, we are ready to prove well-nestedness.

Lemma 9.29. *Every $e \in \mathcal{T}_{\mathsf{SPR}}$ is either a recursive or progressive state in A_{SPR}.*

Consequently, the series-parallel rational syntactic PA is well-nested.

Proof. Let $e \in \mathcal{T}_{\mathsf{SPR}}$ be non-progressive; it suffices to show that e is recursive. By Lemma 9.28, we know that there exists $\phi \in \mathbb{M}(\mathcal{T}_{\mathsf{SPR}})$ such that $\gamma_{\mathsf{SPR}}(e, \phi) \neq \emptyset$, where $\phi = \{\!|f, g^\dagger|\!\}$, with $f \prec_{\mathsf{SPR}} e$. Furthermore, $g^\dagger \prec_{\mathsf{SPR}} e$ does *not* hold (otherwise e would be progressive) and therefore $e \preceq_{\mathsf{SPR}} g^\dagger$ must hold. Since $g^\dagger \preceq_{\mathsf{SPR}} e$ as well, we conclude by Lemma 9.27 that $e = g^\dagger$. Since the conditions on recursive states hold for spr-expressions of the form $g^\dagger$, we conclude that e is recursive. □

In total, we conclude with the expressions-to-automata direction of our extended Kleene theorem.

Theorem 9.30 (Expressions to automata)**.** *For every $e \in \mathcal{T}_{\mathsf{SPR}}$, we can obtain a well-nested and finite PA A with a state q such that $L_A(q) = [\![e]\!]_{\mathsf{SPR}}$.*

Proof. Take the series-parallel rational syntactic pomset automaton A_{SPR} and restrict it to the support of e. By Lemmas 9.15 and 9.24, we then obtain a finite and well-nested pomset automaton $A' = A_{\mathsf{SPR}}[\pi_{\mathsf{SPR}}(e)]$; by Lemma 7.20, we can conclude that $L_{A'}(e') = [\![e]\!]_{\mathsf{SPR}}$. □

9.2.2 Automata to expressions

We now go on to extend our earlier conversion of fork-acyclic and finite PAs to sr-expressions (Theorem 7.41) into a conversion from well-nested and finite PAs to spr-expressions. We first extend the machinery of solutions to pomset automata to spr-expressions, as follows.

Definition 9.31 (Solution of a PA)**.** Let $A = \langle Q, F, \delta, \gamma \rangle$ be a PA, and let $\approx$ be a EBKA congruence on $\mathcal{T}_{\mathsf{SPR}}(\Delta)$ with $\Sigma \subseteq \Delta$. We say $s : Q \to \mathcal{T}_{\mathsf{SPR}}(\Delta)$ is a *$\approx$-solution* to A if, for every $q \in Q$:

$$[q \in F] + \sum_{q' \in \delta(q, \mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \lessapprox s(q)$$

Also, s is a *least $\approx$-solution* to A if for every $\approx$-solution s' we have $s(q) \lessapprox s(q')$ for all $q \in Q$. We call $s : Q \to \mathcal{T}_{\mathsf{SPR}}$ the *least spr-solution* to A if it is the least $\approx$-solution for any EBKA congruence $\approx$.

Example 9.32. Let $A = \langle Q, F, \delta, \gamma \rangle$ be the PA in Figure 9.2, and let $\approx$ be a BKA congruence on $\mathcal{T}(\Delta)$ with $\Sigma \subseteq \Delta$. The constraints on a $\approx$-solution $s : Q \to \mathcal{T}(\Delta)$ to A can then be written as

$$1 + (s(q_1) \parallel s(q_3)) \cdot s(q_2) \lessapprox s(q_1) \qquad 1 \lessapprox s(q_2) \qquad \mathtt{a} \cdot s(q_2) \lessapprox s(q_3)$$

Like the least solution gave us series-rational expressions to describe the languages accepted by a PA, the least spr-solution gives us series-parallel rational expressions that describe the languages accepted by a PA. The proof of this is completely analogous to that of Lemma 7.39.

Lemma 9.33. *Let $A = \langle Q, F, \delta, \gamma \rangle$ be a pomset automaton. If $s : Q \to \mathcal{T}_{\mathsf{SPR}}$ is the least spr-solution to A, then for $q \in Q$ it holds that $L_A(q) = [\![s(q)]\!]_{\mathsf{SPR}}$.*

The idea is that, since spr-expressions are strictly more expressive than sr-expressions, more PAs should be solvable. Sure enough, such a solution exists for finite and well-nested PAs, as follows.

Lemma 9.34. *Let A be a well-nested and finite PA. We can construct the least spr-solution to A.*

Proof. We proceed by induction on the depth of $A = \langle Q, F, \delta, \gamma \rangle$. In the base, where $D_A = 0$, we have $Q = \emptyset$, and thus the claim holds vacuously. For the inductive step, assume that the claim holds for well-nested and finite pomset automata of depth $D_A - 1$. We choose $Q' = \{q' \in Q : D_A(q) < D_A\}$, and note that Q' is support-closed by construction. Let $A' = A[Q']$; it is not hard to see that $D_{A'} < D_A$. By induction, we then obtain $s' : Q' \to \mathcal{T}$ as the least spr-solution to A'.

We now construct an spr-solution $s : Q \to \mathcal{T}_{\mathsf{SPR}}$ to A; to this end, we should choose an spr-expression $s(q)$ for any $q \in Q$. To this end, we divide Q into the following sets:

- Q_{stp}, the states in Q without any outgoing transitions, and
- Q_{ind}, the states in Q' that do have outgoing transitions, and
- Q_{rec}, the recursive states in $Q \setminus Q'$ with outgoing transitions, and
- Q_{pro}, the progressive (and non-recursive) states in $Q \setminus Q'$ with outgoing transitions.

Since A is well-nested, these cover Q', i.e.,, we have $Q = Q_{\mathsf{stp}} \cup Q_{\mathsf{ind}} \cup Q_{\mathsf{rec}} \cup Q_{\mathsf{pro}}$. Moreover, they are disjoint by construction. This allows us to choose $s(q) \in \mathcal{T}_{\mathsf{SPR}}$ on a case-by-case basis, as follows:

- When $q \in Q_{\mathsf{stp}}$, there are no outgoing transitions on q, and hence the constraint on $s(q)$ simplifies into $[q \in F] \lessapprox s(q)$. We can therefore choose $s(q) = [q \in F]$ to satisfy this constraint.
- For $q \in Q_{\mathsf{ind}}$ we choose $s(q) = s'(q)$. To see that this satisfies the constraint on $s(q)$, first suppose $q' \preceq_A q$. Since $q \in Q_{\mathsf{ind}} \subseteq Q'$ and Q' is support-closed, $q' \in Q'$. Hence, if $q' \in Q_{\mathsf{ind}}$, then $s(q') = s'(q')$; otherwise, if $q' \in Q_{\mathsf{stp}}$, then $s(q') = [q \in F] = [q \in F'] \lessapprox s'(q')$, because s' is a solution to A'. Thus, in either case, $s(q') \lessapprox s'(q')$. We can then derive that

$$\begin{aligned}
&[q \in F] + \sum_{q' \in \delta(q, \mathsf{a})} \mathsf{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!| r_1, \dots, r_n |\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\
&\lessapprox [q \in F'] + \sum_{q' \in \delta(q, \mathsf{a})} \mathsf{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!| r_1, \dots, r_n |\!\})} (s'(r_1) \parallel \cdots \parallel s'(r_n)) \cdot s'(q') \\
&\approx [q \in F'] + \sum_{q' \in \delta'(q, \mathsf{a})} \mathsf{a} \cdot s(q') + \sum_{q' \in \gamma'(q, \{\!| r_1, \dots, r_n |\!\})} (s'(r_1) \parallel \cdots \parallel s'(r_n)) \cdot s'(q') \\
&\lessapprox s'(q) = s(q)
\end{aligned}$$

In the second step, we use that if $q \in Q_{\mathsf{ind}} \subseteq Q'$, then for all $\mathtt{a} \in \Sigma$ we have $\delta(q, \mathtt{a}) = \delta'(q, \mathtt{a})$, and for all $\phi \in \mathbb{M}(Q)$ we have $\gamma(q, \phi) = \gamma'(q, \phi)$ when $\phi \in \mathbb{M}(Q')$, and $\gamma(q, \phi) = \emptyset$ otherwise.

- Next, let $q \in Q_{\mathsf{rec}}$. Since q is recursive, it admits two kinds of transitions: (a) sequential and parallel transitions to states in Q', where the fork targets are from Q' as well; and (b) parallel transitions to states in Q_{stp}, where exactly one fork target is q itself, and the others are states from Q'. Hence, we can rewrite the constraint on $s(q)$ for any EBKA congruence $\lessapprox$ into

$$\begin{aligned}[t] [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_1|\!\}) \cap Q'} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\ + \sum_{q' \in \gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}}} (s(q) \parallel s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \lessapprox s(q) \end{aligned} \tag{9.1}$$

Now, since for $q' \in Q_{\mathsf{stp}}$ we have chosen $s(q') = [q' \in F]$, it follows that in the last sum only the terms where $q' \in F$ matter. Thus, eq. (9.1) is equivalent to

$$\begin{aligned}[t] [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_1|\!\}) \cap Q'} (s(r_1) \parallel \cdots \parallel s'(r_n)) \cdot s(q') \\ + \sum_{q' \in \gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}} \cap F} s(q) \parallel s(r_1) \parallel \cdots \parallel s(r_n) \lessapprox s(q) \end{aligned} \tag{9.2}$$

By distributivity of $\parallel$ over $+$, we can show that eq. (9.2) is equivalent to

$$\begin{aligned}[t] [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_1|\!\}) \cap Q'} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\ + s(q) \parallel \Bigg(\sum_{\gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}} \cap F \neq \emptyset} s(r_1) \parallel \cdots \parallel s(r_n) \Bigg) \lessapprox s(q) \end{aligned} \tag{9.3}$$

Using the fixpoint axiom for parallel star, eq. (9.3) implies that $e_q \parallel f_q^\dagger \lessapprox s(q)$, in which

$$e_q = [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot s(q') \qquad f_q = \sum_{\gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}} \cap F \neq \emptyset} s(r_1) \parallel \cdots \parallel s(r_n)$$

Now, because q is recursive, all of the $s(q')$ and $s(r_i)$ in the above are already defined in the previous two steps; we have definite values for e_q and f_q. We choose $s(q) = e_q \parallel f_q^\dagger$ for our putative solution. To see that this indeed satisfies the requirement on $s(q)$, note that $e_q + e_q \parallel f_q^\dagger \parallel f_q \approx e_q \parallel (1 + f_q^\dagger \parallel f_q) \approx e_q \parallel f_q^\dagger = s(q)$. Hence, our choice of $s(q)$ satisfies eq. (9.3), and therefore the constraint on $s(q)$ in eq. (9.1).

- It remains to find candidate values for $q \in Q_{\mathsf{pro}}$. To this end, we create the series-parallel rational system $\mathcal{S} = \langle M, b \rangle$ on Q_{pro}, defined as follows:

$$M(q,q') = \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\}) \cap Q_{\mathsf{pro}}} s(r_1) \parallel \cdots \parallel s(r_n)$$

$$b(q) = [q \in F] + \sum_{q' \in \delta(q,\mathtt{a}) \setminus Q_{\mathsf{pro}}} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\}) \setminus Q_{\mathsf{pro}}} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q')$$

Note that in the above, if $q' \in \gamma(q, \{\!|r_1, \dots, r_n|\!\})$ then $r_1, \dots, r_n \in Q'$, and therefore $s(r_i)$ is defined for $1 \leq i \leq n$; this shows that M is well-defined; similarly, b is well-defined.

Let $s'' : Q_{\mathsf{pro}} \to \mathcal{T}_{\mathsf{SPR}}$ be the least solution to $\mathcal{S}$, obtained through Theorem 9.6. For $q \in Q_{\mathsf{pro}}$, we choose $s(q) = s''(q)$. To see that this satisfies the requirement on $s(q)$, we calculate:

$$\begin{aligned}
& [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\})} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\
\approx\ & [q \in F] + \sum_{q' \in \delta(q,\mathtt{a}) \setminus Q_{\mathsf{pro}}} \mathtt{a} \cdot s(q') + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\}) \setminus Q_{\mathsf{pro}}} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s(q') \\
& + \sum_{q' \in \delta(q,\mathtt{a}) \cap Q_{\mathsf{pro}}} \mathtt{a} \cdot s''(q') + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\}) \cap Q_{\mathsf{pro}}} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s''(q') && \text{(def. } s\text{)} \\
\approx\ & b(q) + \sum_{q' \in \delta(q,\mathtt{a}) \setminus Q_{\mathsf{pro}}} \mathtt{a} \cdot s''(q') + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\}) \cap Q_{\mathsf{pro}}} (s(r_1) \parallel \cdots \parallel s(r_n)) \cdot s''(q') && \text{(def. } b\text{)} \\
\approx\ & b(q) + \sum_{q' \in Q_{\mathsf{pro}}} M(q,q') \cdot s''(q) && \text{(def. } M\text{)} \\
\lessapprox\ & s''(q) = s(q) && (s'' \text{ is a } {\approx}\text{-solution to } \mathcal{S})
\end{aligned}$$

Thus, s satisfies the constraints on every $q \in Q$; hence, s is a solution to A.

To see that it is the *least* solution to A, let $\approx$ be an EBKA congruence, and let $t : Q \to \mathcal{T}_{\mathsf{SPR}}$ be a $\approx$-solution to A. To show that $s(q) \lessapprox t(q)$ for $q \in Q$, we distinguish three cases:

- If $q \in Q_{\mathsf{stp}}$, then $s(q) = [q \in F] \lessapprox t(q)$, since t is a solution to A.
- If $q \in Q_{\mathsf{ind}}$, then let t' be the restriction of t to Q'. We claim that t' is a solution to A'. To see this, we calculate for $q \in Q'$ that

$$\begin{aligned}
& [q \in F] + \sum_{q' \in \delta'(q,\mathtt{a})} \mathtt{a} \cdot t'(q) + \sum_{q' \in \gamma'(q, \{\!|r_1,\dots,r_n|\!\})} (t'(r_1) \parallel \cdots \parallel t'(r_n)) \cdot t'(q') \\
\lessapprox\ & [q \in F] + \sum_{q' \in \delta'(q,\mathtt{a})} \mathtt{a} \cdot t'(q) + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\})} (t'(r_1) \parallel \cdots \parallel t'(r_n)) \cdot t'(q') \\
\approx\ & [q \in F] + \sum_{q' \in \delta'(q,\mathtt{a})} \mathtt{a} \cdot t(q) + \sum_{q' \in \gamma(q, \{\!|r_1,\dots,r_n|\!\})} (t(r_1) \parallel \cdots \parallel t(r_n)) \cdot t(q') \\
\lessapprox\ & t(q) = t'(q)
\end{aligned}$$

Since s' is the *least* $\approx$-solution to A', we have for $q \in Q'$ that $s'(q) \lessapprox t'(q)$. Thus, if $q \in Q_{\mathsf{ind}}$, then we can derive that $s(q) = s'(q) \lessapprox t'(q) = t(q)$.

- If $q \in Q_{\mathsf{rec}}$, then $t(q)$ satisfies the same constraint as $s(q)$ in eq. (9.3); concretely, we have

$$[q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot t(q') + \sum_{q' \in \gamma(q, \{\!|r_1, \dots, r_1|\!\}) \cap Q'} (t(r_1) \parallel \cdots \parallel t(r_n)) \cdot t(q')$$
$$+ t(q) \parallel \Big(\sum_{\gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}} \cap F \neq \emptyset} t(r_1) \parallel \cdots \parallel t(r_n) \Big) \lessapprox t(q) \tag{9.4}$$

By the fixpoint axiom for parallel star, it then follows that $g_q \parallel h_q^\dagger \lessapprox t(q)$, in which

$$g_q = [q \in F] + \sum_{q' \in \delta(q,\mathtt{a})} \mathtt{a} \cdot t(q') \qquad h_q = \sum_{\gamma(q, \{\!|q, r_1, \dots, r_n|\!\}) \cap Q_{\mathsf{stp}} \cap F \neq \emptyset} t(r_1) \parallel \cdots \parallel t(r_n)$$

But, since q is recursive, the q' and r_i' must belong to Q'. Since by the previous cases we have for $q' \in Q'$ that $s(q') \lessapprox t(q')$, it follows that $e_q \lessapprox g_q$ and $f_q \lessapprox h_q$; hence, we derive

$$s(q) = e_q \parallel f_q^\dagger \lessapprox g_q \parallel h_q^\dagger = t(q)$$

- If $q \in Q_{\mathsf{pro}}$, then let t'' be the restriction of t to Q_{pro}. It is not hard to show that t'' is a $\approx$-solution to the series-parallel rational system $\mathcal{S}$. Since s'' is the least $\approx$-solution to $\mathcal{S}$, we have for $q \in Q_{\mathsf{pro}}$ that $s(q) = s''(q) \lessapprox t''(q) = t(q)$. This completes the proof. □

We can then surmise that the languages accepted by any well-nested and finite PA can be described by series-parallel rational expressions, which we record as follows.

Theorem 9.35 (Automata to expressions)**.** *If $A = \langle Q, F, \delta, \gamma \rangle$ is a well-nested and finite PA, then we can construct for every $q \in Q$ an spr-expression $e \in \mathcal{T}_{\mathsf{SPR}}$ such that $L_A(q) = [\![e]\!]_{\mathsf{SPR}}$.*

Proof. Apply Lemma 9.34 to obtain a solution $s : Q \to \mathcal{T}_{\mathsf{SPR}}$ to A. By Lemma 9.33, we know for $q \in Q$ that $L_A(q) = [\![s(q)]\!]_{\mathsf{SPR}}$. Hence, given $q \in Q$, we can choose $e = s(q)$ to satisfy the claim. □

This allows us to conclude this chapter with the desired Kleene theorem that relates series-parallel rational expressions to finite and well-nested pomset automata, as follows.

Corollary 9.36 (Extended Kleene theorem for pomset languages)**.** *Let $L \subseteq \mathsf{SP}$. Then L is series-parallel rational if and only if it is accepted by a finite and well-nested pomset automaton.*

Summary of this chapter Series-rational expressions can be extended to series-parallel rational expressions, which include the "parallel star" operator. We showed that languages expressed by series-parallel rational expressions are precisely the languages accepted by a structurally restricted fragment of PAs. This further generalises the Kleene theorem established in an earlier chapter.

9.A Proofs for expressions-to-automata translation

Lemma 9.15. *Let A be a PA, with Q' be support-closed in A. If A is well-nested, then so is $A[Q']$.*

Proof. Recall from the proof of Lemma 7.20 that for $q, q' \in Q'$ we have that $q \preceq_{A[Q']} q'$ if and only if $q \preceq_A q'$. It is then straightforward to show that for $q \in Q'$, we have that q is recursive (respectively progressive) in A if and only if it is recursive (respectively progressive) in $A[Q']$. Since very state in A is recursive or progressive, it follows that the same holds for $A[Q']$. □

Lemma 9.20. *Let $e \in \mathcal{T}_{\mathsf{SPR}}$ and $U \in \mathsf{SP}$. The following are equivalent:*

(i) There exists $f \in \mathcal{F}$ such that $e^\dagger \xrightarrow{U}_{\mathsf{SPR}} f$.

(ii) $U = U_1 \parallel \cdots \parallel U_n$, such that for $1 \leq i \leq n$ there exists $f_i \in \mathcal{F}_{\mathsf{SPR}}$ with $e \xrightarrow{U_i}_{\mathsf{SR}} f_i$.

Proof. To show that (i) implies (ii), note that if $e^\dagger \xrightarrow{U}_{\mathsf{SPR}} f$, then this run must either be trivial or a parallel unit run. Now, if $e^\dagger \xrightarrow{U}_{\mathsf{SPR}} f$ is trivial, then we can choose $n = 0$ to satisfy the claim. Otherwise, if $e^\dagger \xrightarrow{U}_{\mathsf{SPR}} f$ is a parallel unit run, then $f = 1$ and $U = V \parallel W$ such that $e \xrightarrow{V}_{\mathsf{SPR}} e'$ and $e^\dagger \xrightarrow{W}_{\mathsf{SPR}} f'$ for some $e', f' \in \mathcal{F}_{\mathsf{SPR}}$. By induction, we find that $W = W_1 \parallel \cdots \parallel W_{n'}$ such that for $1 \leq i \leq n'$ there exists $f_i \in \mathcal{F}_{\mathsf{SPR}}$ with $e \xrightarrow{W_i}_{\mathsf{SPR}} f_i$. If we then choose $n = n' + 1$ and set for $1 \leq i < n$ that $U_i = W_i$ and $f_i = f'_i$, as well as $U_n = V$ and $f_i = e'$, then $U = W \parallel V = W_1 \parallel \cdots \parallel W_{n'} \parallel V = U_1 \parallel \cdots \parallel U_n$, and for $1 \leq i \leq n$ we have $e \xrightarrow{U_i}_{\mathsf{SPR}} f_i$.

To show that (ii) implies (i), we proceed by induction on n. In the base, where $n = 0$, we have that $U = 1$; we can choose $f = e^\dagger$ to satisfy the claim. For the inductive step, let $n > 0$ and assume that the claim holds for $n-1$. By induction, we then find an $f' \in \mathcal{F}_{\mathsf{SPR}}$ such that $e^\dagger \xrightarrow{U_1 \parallel \cdots \parallel U_{n-1}}_{\mathsf{SPR}} f'$. Since $1 \in \gamma_{\mathsf{SPR}}(e^\dagger, \{\!|e, e^\dagger|\!\})$, we can then choose $f = 1$ to find that $e^\dagger \xrightarrow{U_1 \parallel \cdots \parallel U_{n-1} \parallel U_n}_{\mathsf{SPR}} f$. □

Lemma 9.21. *Let $e, f \in \mathcal{T}_{\mathsf{SPR}}$. The following hold:*

$$L_{\mathsf{SPR}}(e+f) = L_{\mathsf{SPR}}(e) + L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e \cdot f) = L_{\mathsf{SPR}}(e) \cdot L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e^*) = L_{\mathsf{SPR}}(e)^*$$

$$L_{\mathsf{SPR}}(e \parallel f) = L_{\mathsf{SPR}}(e) \parallel L_{\mathsf{SPR}}(f) \qquad L_{\mathsf{SPR}}(e^\dagger) = L_{\mathsf{SPR}}(e)^\dagger$$

Proof. Since Lemmas 7.25 to 7.28 apply to the series-rational syntactic PA as well, the first four equalities follow by the same argument as in Lemma 7.29. Similarly, the last equality follows the same structure as the arguments in Lemma 7.29, where we apply Lemma 9.20. □

Lemma 9.22. *For all $e \in \mathcal{T}_{\mathsf{SPR}}$, we have $L_{\mathsf{SPR}}(e) = [\![e]\!]_{\mathsf{SPR}}$.*

Proof. The proof proceeds similarly to Lemma 7.30; indeed, the base cases are argued similarly, and the inductive cases follow from Lemma 9.21. □

Lemma 9.24. *For every $e \in \mathcal{T}_{\mathsf{SPR}}$, we have that $e \in R(e)$ and $R(e)$ is support-closed.*

Consequently, the series-parallel rational syntactic PA is bounded.

Proof. The proof proceeds by induction on e. For the base, the same arguments as in Lemma 7.34 apply. In the inductive step, the same holds for all operators except the parallel star, which is new. Thus, let $e = e_1^\dagger$, and suppose the claim holds for e_1. By construction, we have that $e_1^\dagger \in R(e)$. We should show that if $e' \in R(e)$, then the support of e' is included in $R(e)$. If $e' = 1$, this holds immediately, and if $e' \in R(e_1)$, then the claim follows by induction. It remains to consider the case where $e' = e_1^\dagger$. As in Lemma 7.34, it suffices to show that if $f \preceq_{\mathsf{SPR}} e_1^\dagger$ is a consequence of one of the rules generating $\preceq_{\mathsf{SPR}}$, then $f \in R(e)$. This gives us three cases to consider.

- The case where $f \in \delta_{\mathsf{SPR}}(e_1^\dagger, \mathtt{a})$ for some $\mathtt{a} \in \Sigma$ can be disregarded, for $\delta_{\mathsf{SPR}}(e_1^\dagger, \mathtt{a}) = \emptyset$.
- If $f \in \gamma_{\mathsf{SPR}}(e_1^\dagger, \phi)$, then $f = 1$ by construction of γ_{SPR}. Hence $f \in R(e)$.
- If $f \in \phi \in \mathbb{M}(\mathcal{T})$ and $\gamma_{\mathsf{SPR}}(e_1^\dagger, \phi) \neq \emptyset$, then either $f = e_1^\dagger$ or $f = e_1$. In the former case, $f \in R(e)$ by definition; in the latter case, $f \in R(e_1) \subseteq R(e)$ by induction. □

Lemma 9.26. *For $e, f \in \mathcal{T}_{\mathsf{SPR}}$ such that $e \preceq_{\mathsf{SPR}} f$, we have $\mathsf{d}_{\parallel}(e) \leq \mathsf{d}_{\parallel}(f)$ and $\mathsf{d}_{\dagger}(e) \leq \mathsf{d}_{\dagger}(f)$.*

Proof. It suffices to verify the claim for the pairs that generate $\preceq_{\mathsf{SPR}}$. This gives us three cases.

- If $e \preceq_{\mathsf{SPR}} f$ because there exists an $\mathtt{a} \in \Sigma$ such that $e \in \delta_{\mathsf{SPR}}(f, \mathtt{a})$, then we proceed by induction on f. In the base, necessarily $f \in \mathtt{a}$ and $e = 1$; but then $\mathsf{d}_{\parallel}(e) = \mathsf{d}_{\dagger}(e) = 0 \leq \mathsf{d}_{\parallel}(f), \mathsf{d}_{\dagger}(f)$ immediately. For the inductive step, we consider only the case where $f = f_1^\dagger$; all other cases follow by an argument similar to the one in the proof of Lemma 7.31. This case can be disregarded, for $\delta_{\mathsf{SPR}}(f_1^\dagger, \mathtt{a}) = \emptyset$ by definition of δ_{SPR}.
- If $e \preceq_{\mathsf{SPR}} f$ because there exists a $\phi \in \mathbb{M}(\mathcal{T})$ such that $e \gamma_{\mathsf{SPR}}(f, \phi)$, then we proceed by induction on f, then the claim follows by an argument similar to the previous case.
- If $e \preceq_{\mathsf{SPR}} f$ because $e \in \phi \in \mathbb{M}(\mathcal{T})$ and $\gamma_{\mathsf{SPR}}(f, \phi) \neq \emptyset$, then we proceed by induction on f. In the base, where $f \in \Sigma \cup \{0, 1\}$, the claim holds vacuously, because $\gamma_{\mathsf{SPR}}(f, \phi) = \emptyset$ for all $\phi \in \mathbb{M}(\mathcal{T})$. In the inductive step, we consider only the case where $f = f_1^\dagger$; all other cases can be treated with an argument similar to the one in the proof of Lemma 7.31. We then find that either $e = f_1$ or $e = f_1^\dagger$; in either case, $\mathsf{d}_{\parallel}(e) \leq \mathsf{d}_{\parallel}(f_1) = \mathsf{d}_{\parallel}(f)$ and $\mathsf{d}_{\dagger}(e) \leq \mathsf{d}_{\dagger}(f)$. □

Lemma 9.28. *Let $e \in \mathcal{T}_{\mathsf{SPR}}$ and $\phi \in \mathbb{M}(\mathcal{T}_{\mathsf{SPR}})$ such that $\gamma_{\mathsf{SPR}}(e, \phi) \neq \emptyset$. Then $\phi = \{\!| f, g |\!\}$ such that either (i) $f \prec_{\mathsf{SPR}} e$ as well as $g \prec_{\mathsf{SPR}} e$, or (ii) $f \prec_{\mathsf{SPR}} e$ and $g = h^\dagger$ for some $h \in \mathcal{T}_{\mathsf{SPR}}$.*

Proof. We proceed by induction on e. In the base, either $e = 0$, $e = 1$, or $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$. In that case, the claim holds vacuously, since for all $\phi \in \mathbb{M}(\mathcal{T}_{\mathsf{SPR}})$ we have that $\gamma_{\mathsf{SPR}}(e, \phi) = \emptyset$. For the inductive step, we have five cases to consider.

- If $e = e_1 + e_2$ or $e = e_1 \cdot e_2$, then either $\gamma_{\mathsf{SPR}}(e_1, \phi) \neq \emptyset$ or $\gamma_{\mathsf{SPR}}(e_2, \phi) \neq \emptyset$. Similarly, when $e = e_1^*$, we have $\gamma_{\mathsf{SPR}}(e_1, \phi) \neq \emptyset$. In any of these cases, the claim follows by induction.
- If $e = e_1 \parallel e_2$, then $\phi = \{\!| e_1, e_2 |\!\}$; in that case $\mathsf{d}_{\parallel}(e_1), \mathsf{d}_{\parallel}(e_2) < \mathsf{d}_{\parallel}(e)$, and hence $e_1, e_2 \prec_{\mathsf{SPR}} e$, which means that the first possibility is satisfied.
- If $e = e_1^\dagger$, then $\phi = \{\!| e_1, e_1^\dagger |\!\}$. Since $\mathsf{d}_{\parallel}(e_1) < \mathsf{d}_{\parallel}(e_1^\dagger)$, we have that $e_1 \prec_{\mathsf{SPR}} e$ by Lemma 9.26. This means that the second possibility is satisfied. □

Further Work

We conclude the second half of this thesis by listing questions that arise from the results in the previous three chapters, accompanied by some preliminary observations.

Coalgebra Universal coalgebra [Rut00] provides a uniform toolset to develop operational models; we wonder whether it can be applied to pomset languages, too. In such an approach, an operational description of a pomset language would be a coalgebra for a fixed functor F, and the semantics in terms of pomsets would arise from the unique F-coalgebra homomorphism into the final F-coalgebra. The added value of such an approach would be that it gives a natural way to compare coalgebras for language equivalence using a notion of bisimulation arising from the functor F [Rut98]. The decision procedure arising from this can then be optimised, guided by the functor [BP13; RBR13].

For such an approach to work, one would have to find a functor F such that the set of (series-parallel) pomsets can be equipped with an F-coalgebra structure in some natural way. However, the functors that we tried to use in modelling pomset languages did not admit a final coalgebra for cardinality reasons, as a consequence of Lambek's lemma [Lam68]. Another approach could be to try and describe pomset automata coalgebraically; the problem here is that the non-local nature of the parallel transition function also has no obvious coalgebraic description. Further work could try to tackle this problem by looking at functors in categories with more structure.

Completeness The proof that $\equiv$ is complete w.r.t. $[\![-]\!]$ does not involve any operational description of pomset languages [LS14]. We would like to find out if this result can be obtained through pomset automata, instead. A proof like this would first represent language-equivalent sr-expressions as finite and fork-acyclic pomset automata having those expressions as solutions. The second step would be to use language equivalence of those pomset automata to argue that they must have the same solution. This technique can be seen as the core of Kozen's completeness proof of completeness for Kleene algebra [Koz94], and also appears in more recent coalgebraic accounts [Jac06].

Reduction The construction that allowed us to reduce the hypotheses in exch to the empty set of hypotheses in Chapter 5 was phrased in terms of finding a solution to a series-rational system for each level of nesting of parallel composition, i.e., we had to solve a series-rational system at each level. We would like to return to this constriction and phrase it in terms of solutions to fork-acyclic and finite pomset automata, in effect hiding the nested solving of series-rational systems in an appeal to Lemma 7.40. The added benefit of this is that it would also give us another (possibly more efficient) way to check equivalence w.r.t. $\equiv^{\mathsf{exch}}$, by comparing the pomset automata.

Systems At multiple points, we have constructed an sr-expression or spr-expression by constructing a system of equations, and invoking a result that allowed us to obtain a solution to that system. In a sense, our Kleene theorems tell us that we have reached the limits of this formalism: every series-rational system is the solution of some finite and fork-acyclic pomset automaton, and likewise every series-parallel rational system is the solution of some finite and well-nested pomset automaton. Checking fork-acyclicity or well-nestedness, however, is not always easy. It would be helpful to create an even more generic formats for systems that solve to sr-expressions or spr-expressions.

Expressions There exists another class of pomset languages, called *recognisable* pomset languages [LW00], which is strictly more general than series-parallel rational pomset languages. More specifically, languages in this class can be described by a morphism into a finite bi-monoid, and may have unbounded depth. We are curious which class of pomset automata this class of pomset languages would correspond to. Additionally, it would be interesting to come up with an extension of spr-expressions whose semantics would correspond exactly to recognisable pomset languages, and try to find out if equivalence of these expressions can be axiomatised as well.

Complexity Our investigation in Chapter 8 focused on showing that equivalence of pomset automata is decidable. The repeated use of the inverse powerset construction [BT14] in our procedure suggests that it may not be very efficient. It has also been shown that deciding equivalence of sr-expressions is EXPSPACE-complete [BPS17], which furthermore suggests that deciding equivalence of pomset automata is a hard problem. Further analysis is necessary to find out a tight complexity bound on our procedure, especially with regard to how much effort goes into the conversion to a well-structured pomset automaton. Experience has shown that while equivalence of non-deterministic finite automata is PSPACE-complete in general, there are still procedures that can practically be used to decide equivalence [BP13]; the same might be true for our algorithm. We would like to implement and benchmark our algorithm to get a feeling for its efficiency.

Decidability In Chapter 8, we showed that language equivalence of finite and fork-acyclic pomset automata is decidable, thereby giving a novel proof of decidability of equivalence for sr-expressions. Equivalence of spr-expressions is also be decidable [LS14], but no algorithm based on a operational perspective is known. One way to obtain such a proof could be to generalise the decision procedure for fork-acyclic and finite pomset automata to well-nested and finite pomset automata. Presumably, we would first need to show how this procedure would work for well-structured, well-nested and finite pomset automata, before showing how we can guarantee well-structuredness.

Context-free languages In Chapter 9, we showed that any context-free language can be implemented by a pomset automaton with unary forks, and remarked that the converse would not be too hard to show. A somewhat tangential but still interesting followup to this correspondence would be to investigate the connection between context-free languages and unary pomset automata even further. For instance, is there a class of unary pomset automata that corresponds to deterministic context free languages? Since equivalence in this class of languages is decidable [Sén01], we wonder whether this putative class of unary pomset automata may give rise to a novel decision procedure.

Bisimilarity A common way to compare operational representations of behaviour is to check whether they are *bisimilar*, i.e., whether computational steps in one such representation can be mimicked by the other, and vice versa. It is not hard to derive a notion of bisimilarity for pomset automata, and to show that bisimilarity is sufficient (but not necessary) for language equivalence. Bisimilarity of pomset automata might be a good preliminary check for equivalence — if two states are bisimilar, then their languages coincide, and thus a (possibly resource-intensive) algorithm to compare languages need not be called. For this to work, one would need to derive a procedure for checking bisimilarity in pomset automata, analyse its complexity, and benchmark it.

Bibliography

[AF06] Luca Aceto and Wan Fokkink. "The Quest for Equational Axiomatizations of Parallel Composition: Status and Open Problems". In: *Electron. Notes Theor. Comput. Sci.* 162 (2006), pp. 43–48. DOI: 10.1016/j.entcs.2005.12.076.

[AFG$^+$14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. "NetKAT: semantic foundations for networks". In: *POPL.* 2014, pp. 113–126. DOI: 10.1145/2535838.2535862.

[AI07] Luca Aceto and Anna Ingólfsdóttir. "The Saga of the Axiomatization of Parallel Composition". In: *CONCUR.* 2007, pp. 2–16. DOI: 10.1007/978-3-540-74407-8_2.

[AK01] Allegra Angus and Dexter Kozen. *Kleene Algebra with Tests and Program Schematology.* Tech. rep. TR2001-1844. Cornell University, July 2001. HANDLE: 1813/5831.

[AMS$^+$11] Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. "Litmus: Running Tests against Hardware". In: *TACAS.* 2011, pp. 41–44. DOI: 10.1007/978-3-642-19835-9_5.

[Ant96] Valentin M. Antimirov. "Partial Derivatives of Regular Expressions and Finite Automaton Constructions". In: *Theor. Comput. Sci.* 155.2 (1996), pp. 291–319. DOI: 10.1016/0304-3975(95)00182-4.

[Ard61] Dean N. Arden. "Delayed-logic and finite-state machines". In: *SWCT.* 1961, pp. 133–151. DOI: 10.1109/FOCS.1961.13.

[Bac75] Roland Backhouse. "Closure algorithms and the star-height problem of regular languages". PhD thesis. University of London, 1975. HANDLE: 10044/1/22243.

[BB70] Garrett Birkhoff and Thomas C. Bartee. *Modern applied algebra.* McGraw-Hill, 1970. ISBN: 978-0070053816.

[BD74] Raymond Balbes and Philip Dwinger. *Distributive lattices.* University of Missouri Press, 1974. ISBN: 978-0983801108.

[BÉ96] Stephen L. Bloom and Zoltán Ésik. "Free Shuffle Algebras in Language Varieties". In: *Theor. Comput. Sci.* 163.1&2 (1996), pp. 55–98. DOI: 10.1016/0304-3975(95)00230-8.

[Bir48] Garrett Birkhoff. *Lattice Theory.* Colloquium publications. American Mathematical Society, 1948. ISBN: 9780821889534.

[BK02] Adam Barth and Dexter Kozen. *Equational Verification of Cache Blocking in LU Decomposition using Kleene Algebra with Tests.* Tech. rep. TR2002-1865. Cornell University, June 2002. HANDLE: 1813/5848.

[BK87] Jan Bergstra and Jan Willem Klop. *ACP_τ: a universal axiom system for process specification.* Tech. rep. Mathematisch Centrum, 1987. CWI: 6132.

[BLB19] Astrid Belder, Bas Luttik, and Jos C. M. Baeten. "Sequencing and Intermediate Acceptance: Axiomatisation and Decidability of Bisimilarity". In: *CALCO.* 2019, 11:1–11:22. DOI: 10.4230/LIPIcs.CALCO.2019.11.

[Bof90] Maurice Boffa. "Une remarque sur les systèmes complets d'identités rationnelles". In: *ITA* 24 (1990), pp. 419–428.

[BP13] Filippo Bonchi and Damien Pous. "Checking NFA equivalence with bisimulations up to congruence". In: *POPL.* 2013, pp. 457–468. DOI: 10.1145/2429069.2429124.

[BP17] Paul Brunet and Damien Pous. "Petri Automata". In: *Logical Methods in Computer Science* 13.3 (2017). DOI: 10.23638/LMCS-13(3:33)2017.

[BP20] Paul Brunet and David J. Pym. "Pomsets with Boxes: Protection, Separation, and Locality in Concurrent Kleene Algebra". In: *FSCD.* 2020, 8:1–8:16. DOI: 10.4230/LIPIcs.FSCD.2020.8.

[BPS17] Paul Brunet, Damien Pous, and Georg Struth. "On Decidability of Concurrent Kleene Algebra". In: *CONCUR.* 2017, 28:1–28:15. DOI: 10.4230/LIPIcs.CONCUR.2017.28.

[BPS61] Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. "On formal properties of simple phrase structure grammars". In: *Sprachtypologie und Universalienforschung* 14 (1961), pp. 143–172.

[Bro07] Stephen Brookes. "A semantics for concurrent separation logic". In: *Theor. Comput. Sci.* 375.1-3 (2007), pp. 227–270. DOI: 10.1016/j.tcs.2006.12.034.

[Brz64] Janusz A. Brzozowski. “Derivatives of Regular Expressions”. In: *J. ACM* 11.4 (1964), pp. 481–494. DOI: `10.1145/321239.321249`.

[BT14] Janusz A. Brzozowski and Hellis Tamm. “Theory of átomata”. In: *Theor. Comput. Sci.* 539 (2014), pp. 13–27. DOI: `10.1016/j.tcs.2014.04.016`.

[CHM17] Robert J. Colvin, Ian J. Hayes, and Larissa A. Meinicke. “Designing a semantic model for a wide-spectrum language with concurrency”. In: *Formal Asp. Comput.* 29.5 (2017), pp. 853–875. DOI: `10.1007/s00165-017-0416-4`.

[CKS96] Ernie Cohen, Dexter Kozen, and Frederick Smith. *The Complexity of Kleene Algebra with Tests.* Tech. rep. TR96-1598. Cornell University, July 1996. HANDLE: `1813/7253`.

[Coh94] Ernie Cohen. *Hypotheses in Kleene Algebra.* Tech. rep. Bellcore, 1994.

[Con71] John Horton Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, Ltd., London, 1971.

[DKP$^+$19] Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. “Kleene Algebra with Hypotheses”. In: *FOSSACS.* 2019, pp. 207–223. DOI: `10.1007/978-3-030-17127-8_12`.

[EH62] Beno Eckmann and Peter Hilton. “Group-like structures in general categories. I. Multiplications and comultiplications”. In: *Mathematische Annalen* 145.3 (1962), pp. 227–255. DOI: `10.1007/bf01451367`.

[ÉN04] Zoltán Ésik and Zoltán L. Németh. “Higher Dimensional Automata”. In: *Journal of Automata, Languages and Combinatorics* 9.1 (2004), pp. 3–29. DOI: `10.25596/jalc-2004-003`.

[Ési02] Zoltán Ésik. “Axiomatizing the subsumption and subword preorders on finite and infinite partial words”. In: *Theor. Comput. Sci.* 273.1-2 (2002), pp. 225–248. DOI: `10.1016/S0304-3975(00)00442-4`.

[FJS$^+$20] Uli Fahrenberg, Christian Johansen, Georg Struth, and Ratan Bahadur Thapa. “Generating Posets Beyond N”. In: *RAMiCS.* 2020, pp. 82–99. DOI: `10.1007/978-3-030-43520-2_6`.

[FKM$^+$15] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. “A Coalgebraic Decision Procedure for NetKAT”. In: *POPL.* 2015, pp. 343–355. DOI: `10.1145/2676726.2677011`.

[Flo67] Robert W. Floyd. “Assigning Meanings to Programs”. In: *Applied Mathematics* 19 (1967), pp. 19–32.

[Gis88] Jay L. Gischer. “The Equational Theory of Pomsets”. In: *Theor. Comput. Sci.* 61 (1988), pp. 199–224. DOI: `10.1016/0304-3975(88)90124-7`.

[Gra81] Jan Grabowski. “On partial languages”. In: *Fundam. Inform.* 4.2 (1981), p. 427.

[HHH+87] Tony Hoare, Ian J. Hayes, Jifeng He, Carroll Morgan, A. W. Roscoe, Jeff W. Sanders, Ib Holm Sørensen, J. Michael Spivey, and Bernard Sufrin. “Laws of Programming”. In: *Commun. ACM* 30.8 (1987), pp. 672–686. DOI: `10.1145/27651.27653`.

[HK71] John E. Hopcroft and Richard M. Karp. *A linear algorithm for testing equivalence of finite automata.* Tech. rep. TR71-114. Dec. 1971.

[HMS+09] Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. “Concurrent Kleene Algebra”. In: *CONCUR*. 2009, pp. 399–414. DOI: `10.1007/978-3-642-04081-8_27`.

[Hoa69] Tony Hoare. “An Axiomatic Basis for Computer Programming”. In: *Commun. ACM* 12.10 (1969), pp. 576–580. DOI: `10.1145/363235.363259`.

[Hoa78] Tony Hoare. “Communicating Sequential Processes”. In: *Commun. ACM* 21.8 (1978), pp. 666–677. DOI: `10.1145/359576.359585`.

[HSM+16] Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. “Developments in Concurrent Kleene Algebra”. In: *J. Log. Algebr. Meth. Program.* 85.4 (2016), pp. 617–636. DOI: `10.1016/j.jlamp.2015.09.012`.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979. ISBN: `978-0201029888`.

[Jac06] Bart Jacobs. “A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages”. In: *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday.* 2006, pp. 375–404. DOI: `10.1007/11780274_20`.

[JM16] Peter Jipsen and M. Andrew Moshier. “Concurrent Kleene Algebra with tests and branching automata”. In: *J. Log. Algebr. Meth. Program.* 85.4 (2016), pp. 637–652. DOI: `10.1016/j.jlamp.2015.12.005`.

[KBL+17] Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. “Brzozowski Goes Concurrent — A Kleene Theorem for Pomset Languages”. In: *CONCUR*. Sept. 2017. DOI: `10.4230/LIPIcs.CONCUR.2017.21`.

[KBL+18] Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. *Equivalence checking for weak bi-Kleene algebra.* Under submission; extends [KBL+17]; revised March 2020. 2018. ARXIV: `1807.02102`.

[KBL+19] Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. "On series-parallel pomset languages: Rationality, context-freeness and automata". In: *J. Log. Algebr. Meth. Program.* 103 (2019). Extends [KBL+17], pp. 130–153. DOI: `10.1016/j.jlamp.2018.12.001`.

[KBR+19] Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. "Kleene Algebra with Observations". In: *CONCUR.* Sept. 2019. DOI: `10.4230/LIPIcs.CONCUR.2019.41`.

[KBS+18] Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. "Concurrent Kleene Algebra: Free Model and Completeness". In: *ESOP.* 2018, pp. 856–882. DOI: `10.1007/978-3-319-89884-1_30`.

[KBS+20] Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. "Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness". In: *FOSSACS.* 2020, pp. 381–400. DOI: `10.1007/978-3-030-45231-5_20`.

[Kle56] Stephen C. Kleene. "Representation of Events in Nerve Nets and Finite Automata". In: *Automata Studies.* Ed. by Claude E. Shannon and John McCarthy. Princeton University Press, 1956, pp. 3–41. ISBN: `978-0691079165`.

[KM14] Dexter Kozen and Konstantinos Mamouras. "Kleene Algebra with Equations". In: *ICALP.* 2014, pp. 280–292. DOI: `10.1007/978-3-662-43951-7_24`.

[Koz00] Dexter Kozen. "On Hoare logic and Kleene algebra with tests". In: *ACM Trans. Comput. Log.* 1.1 (2000), pp. 60–76. DOI: `10.1145/343369.343378`.

[Koz02] Dexter Kozen. "On the Complexity of Reasoning in Kleene Algebra". In: *Inf. Comput.* 179.2 (2002), pp. 152–162. DOI: `10.1006/inco.2001.2960`.

[Koz94] Dexter Kozen. "A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events". In: *Inf. Comput.* 110.2 (1994), pp. 366–390. DOI: `10.1006/inco.1994.1037`.

[Koz96] Dexter Kozen. "Kleene algebra with tests and commutativity conditions". In: *TACAS.* 1996, pp. 14–33. DOI: `10.1007/3-540-61042-1_35`.

[Koz97] Dexter Kozen. "Kleene Algebra with Tests". In: *ACM Trans. Program. Lang. Syst.* 19.3 (1997), pp. 427–443. DOI: `10.1145/256167.256195`.

[KP00] Dexter Kozen and Maria-Christina Patron. "Certification of Compiler Optimizations Using Kleene Algebra with Tests". In: *CL*. 2000, pp. 568–582. DOI: `10.1007/3-540-44957-4_38`.

[Kro90] Daniel Krob. "A Complete System of B-Rational Identities". In: *ICALP*. 1990, pp. 60–73. DOI: `10.1007/BFb0032022`.

[KS96] Dexter Kozen and Frederick Smith. "Kleene Algebra with Tests: Completeness and Decidability". In: *CSL*. 1996, pp. 244–259. DOI: `10.1007/3-540-63172-0_43`.

[KT08] Dexter Kozen and Wei-Lung (Dustin) Tseng. "The Böhm-Jacopini Theorem is False, Propositionally". In: *MPC*. 2008, pp. 177–192. DOI: `10.1007/978-3-540-70594-9_11`.

[Kur22] Casimir Kuratowski. "Sur l'opération $\overline{\mathrm{A}}$ de l'Analysis Situs". In: *Fundamenta Mathematicae* 3.1 (1922), pp. 182–199. ISSN: 0016-2736. DOI: `10.4064/fm-3-1-182-199`.

[Lam68] Joachim Lambek. "A fixpoint theorem for complete categories". In: *Mathematische Zeitschrift* 103.2 (1968), pp. 151–161. ISSN: 1432-1823. DOI: `10.1007/BF01110627`.

[Law75] Eugene L. Lawler. *Optimal sequencing of jobs subject to series parallel precedence constraints.* Tech. rep. Mathematisch Centrum, 1975. CWI: `9714`.

[Law78] E.L. Lawler. "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints". In: *Algorithmic Aspects of Combinatorics.* Vol. 2. Annals of Discrete Mathematics. 1978, pp. 75–90. DOI: `10.1016/S0167-5060(08)70323-6`.

[Lev44] Friedrich W. Levi. "On semigroups". In: *Bull. Calcutta Math. Soc* 36.82 (1944), pp. 141–146. ISSN: 0008-0659.

[LRR03] Kamal Lodaya, D. Ranganayakulu, and K. Rangarajan. "Hierarchical Structure of 1-Safe Petri Nets". In: *ASIAN, Programming Languages and Distributed Computation.* 2003, pp. 173–187. DOI: `10.1007/978-3-540-40965-6_12`.

[LS14] Michael R. Laurence and Georg Struth. "Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages". In: *RAMiCS*. 2014, pp. 65–82. DOI: `10.1007/978-3-319-06251-8_5`.

[LS17] Michael R. Laurence and Georg Struth. *Completeness Theorems for Pomset Languages and Concurrent Kleene Algebras.* 2017. ARXIV: `1705.05896`.

[LW00] Kamal Lodaya and Pascal Weil. "Series-parallel languages and the bounded-width property". In: *Theor. Comp. Sci.* 237.1 (2000), pp. 347–380. DOI: `10.1016/S0304-3975(00)00031-1`.

[Mac98] Saunders Mac Lane. *Categories for the working mathematician.* Springer, 1998. ISBN: `978-1475747218`.

[Mam15] Konstantinos Mamouras. "Extensions of Kleene algebra for program verification". PhD thesis. Ithaca, NY: Cornell University, 2015. HANDLE: `1813/40960`.

[MH15] Bernhard Möller and Tony Hoare. "Exploring an Interface Model for CKA". In: *MPC*. 2015, pp. 1–29. DOI: `10.1007/978-3-319-19797-5_1`.

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: `978-3540102359`. DOI: `10.1007/3-540-10235-3`.

[MW05] Annabelle McIver and Tjark Weber. "Towards Automated Proof Support for Probabilistic Distributed Systems". In: *LPAR*. 2005, pp. 534–548. DOI: `10.1007/11591191_37`.

[MY60] Robert McNaughton and Hisao Yamada. "Regular Expressions and State Graphs for Automata". In: *IRE Trans. Electronic Computers* 9.1 (1960), pp. 39–47. DOI: `10.1109/TEC.1960.5221603`.

[Ner58] Anil Nerode. "Linear Automaton Transformations". In: *Proc. American Mathematical Society* 9.4 (1958), pp. 541–544. DOI: `doi:10.2307/2033204`.

[OHe07] Peter W. O'Hearn. "Resources, concurrency, and local reasoning". In: *Theor. Comput. Sci.* 375.1-3 (2007), pp. 271–307. DOI: `10.1016/j.tcs.2006.12.035`.

[Pet62] Carl Adam Petri. "Kommunikation mit Automaten". PhD thesis. Technische Hochschule Darmstadt, 1962.

[Pil70] Donald L. Pilling. "The Algebra of Operators for Regular Events". PhD thesis. Cambridge University, 1970.

[Pos46] Emil L. Post. "A variant of a recursively unsolvable problem". In: *Bull. Amer. Math. Soc.* 52 (1946), pp. 264–268. DOI: `10.1090/S0002-9904-1946-08555-9`.

[Pra82] Vaughan R. Pratt. "On the Composition of Processes". In: *POPL*. 1982, pp. 213–223. DOI: `10.1145/582153.582177`.

[Pri10] Cristian Prisacariu. "Synchronous Kleene Algebra". In: *J. Log. Algebr. Program.* 79.7 (2010), pp. 608–635. DOI: `10.1016/j.jlap.2010.07.009`.

[RBR13] Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. “Coalgebraic Bisimulation-Up-To”. In: *SOFSEM*. 2013, pp. 369–381. DOI: 10.1007/978-3-642-35843-2_32.

[Red64a] Vladimir N. Red’ko. “On Defining Relations for the Algebra of Regular Events”. In: *Ukrainian Mathematical Journal* 16 (1964). (in Russian), pp. 120–126.

[Red64b] Vladimir N. Red’ko. “On the Algebra of Commutative Events”. In: *Ukrainian Mathematical Journal* 16 (1964). (in Russian), pp. 185–195.

[RS59] Michael O. Rabin and Dana S. Scott. “Finite Automata and Their Decision Problems”. In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125. DOI: 10.1147/rd.32.0114.

[Rut00] Jan J. M. M. Rutten. “Universal coalgebra: a theory of systems”. In: *Theor. Comput. Sci.* 249.1 (2000), pp. 3–80. DOI: 10.1016/S0304-3975(00)00056-6.

[Rut98] Jan J. M. M. Rutten. “Automata and Coinduction (An Exercise in Coalgebra)”. In: *CONCUR*. 1998, pp. 194–218. DOI: 10.1007/BFb0055624.

[Sal66] Arto Salomaa. “Two Complete Axiom Systems for the Algebra of Regular Events”. In: *J. ACM* 13.1 (1966), pp. 158–169. DOI: 10.1145/321312.321326.

[Sén01] Géraud Sénizergues. “$L(A) = L(B)$? Decidability results from complete formal systems”. In: *Theor. Comput. Sci.* 251.1-2 (2001), pp. 1–166. DOI: 10.1016/S0304-3975(00)00285-1.

[SFH$^+$20] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. “Guarded Kleene Algebra with Tests: Verification of Uninterpreted Programs in Nearly Linear Time”. In: *POPL*. Jan. 2020. DOI: 10.1145/3371129.

[Sil10] Alexandra Silva. “Kleene Coalgebra”. PhD thesis. Radboud Universiteit Nijmegen, 2010.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. “Word Problems Requiring Exponential Time: Preliminary Report”. In: *STOC*. 1973, pp. 1–9. DOI: 10.1145/800125.804029.

[Tar75] Robert Endre Tarjan. “Efficiency of a Good But Not Linear Set Union Algorithm”. In: *J. ACM* 22.2 (1975), pp. 215–225. DOI: 10.1145/321879.321884.

[Tho68] Ken Thompson. “Regular Expression Search Algorithm”. In: *Commun. ACM* 11.6 (1968), pp. 419–422. DOI: 10.1145/363347.363387.

[Tsc94] Steven T. Tschantz. “Languages under Concatenation and Shuffling”. In: *Math. Struct. Comput. Sci.* 4.4 (1994), pp. 505–511. DOI: `10.1017/S0960129500000578`.

[Tur37] Alan M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society.* 2nd ser. 42.1 (1937), pp. 230–265. DOI: `10.1112/plms/s2-42.1.230`.

[VTL82] Jacobo Valdes, Robert Endre Tarjan, and Eugene L. Lawler. “The Recognition of Series Parallel Digraphs”. In: *SIAM J. Comput.* 11.2 (1982), pp. 298–313. DOI: `10.1137/0211023`.

[Wat93] Bruce W. Watson. *A taxonomy of finite automata construction algorithms.* Tech. rep. Technische Universiteit Eindhoven, 1993. TU/E: `2482472/9313452`.

[WBD$^+$20] Jana Wagemaker, Paul Brunet, Simon Docherty, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. “Partially Observable Concurrent Kleene Algebra”. In: *CONCUR.* To appear. 2020. ARXIV: `2007.07593`.

[WBK$^+$19] Jana Wagemaker, Marcello M. Bonsangue, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. “Completeness and Incompleteness of Synchronous Kleene Algebra”. In: *MPC.* 2019, pp. 385–413. DOI: `10.1007/978-3-030-33636-3_14`.

[WDH$^+$06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. “Antichains: A New Algorithm for Checking Universality of Finite Automata”. In: *CAV.* 2006, pp. 17–30. DOI: `10.1007/11817963_5`.

Index