Brzozowski Goes Concurrent

A Kleene Theorem for Pomset Languages

T. Kappé[1], P. Brunet[1], B. Luttik[2], A. Silva[1], and F. Zanasi[1]

1 University College London, London, United Kingdom
2 Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

Concurrent Kleene Algebra (CKA) is a mathematical formalism to study programs that exhibit concurrent behaviour. As with previous extensions of Kleene Algebra, characterizing the free model is crucial in order to develop the foundations of the theory and potential applications. For CKA, this has been an open question for a few years and this paper makes an important step towards an answer. We present a new automaton model and a Kleene-like theorem for pomset languages, which are a natural candidate for the free model. There are two substantial differences with previous work: from expressions to automata, we use Brzozowski derivatives, which enable a direct construction of the automaton; from automata to expressions, we provide a syntactic characterization of the automata that denote valid CKA behaviours.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

In their CONCUR'09 paper [4], Hoare, Möller, Struth, and Wehrman introduced Concurrent Kleene Algebra (CKA) as a suitable mathematical framework to study concurrent programs, in the hope of achieving the same elegance that Kozen did when using Kleene Algebra (and extensions) to provide a verification platform for sequential programs.

CKA is a seemingly simple extension of Kleene Algebra (KA): it adds a parallel operator that allows to compositionally specify concurrent behaviours. However, extending the existing KA toolkit — importantly, completeness and decidability results — turns out to be challenging. A fundamental missing ingredient is a characterization of the free model for CKA. This is in striking contrast with KA, where these topics are well understood. Several authors [5, 7] have conjectured the free model to be pomset languages — a generalization of regular languages to sets of partially ordered words.

In KA, Kleene's theorem provided a pillar for developing the toolkit and axiomatization [12], and, by extension, characterizing the free model. In this light, we pursue a Kleene Theorem for CKA. Specifically, we study CKA expressions, also called series-rational expressions, with a denotational model given in terms of pomset languages. Our main contribution is a Kleene Theorem for series-rational expressions, based on two constructions faithfully translating between the denotational model and a newly defined operational model, which we call *pomset automata*. In a nutshell, these are finite-state automata in which computations from a certain state s may branch into parallel threads that contribute to the language of s whenever they both reach a final state.

We are not the first to attempt such a Kleene theorem. However, earlier works [15, 7] fall short of giving a precise correspondence between the denotational and operational models, due to the lack of a suitable automata restriction ensuring that only valid behaviours are accepted. We overcome this situation by introducing a generalization of Brzozowski derivatives [2] in the translation from expressions to automata. This guides us to a *syntactic* restriction

© Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva and Fabio Zanasi;
licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on automata (rather than the *semantic* condition put forward in previous works), which guarantees the existence of a reverse construction, from automata to expressions. Moreover, following the Brzozowski route allows us to bypass a Thompson-like construction [17], avoiding the introduction of ϵ-transitions and non-determinism present in the aforementioned works.

The remainder of this paper is organized as follows. In Section 2, we introduce the necessary notation. In Section 3, we introduce our automaton model as well as some notable subclasses of automata. In Section 4, we discuss how to translate a series-rational expression to a semantically equivalent pomset automaton, while in Section 5 we show how to translate a suitably restricted class of pomset automata to series-rational expressions. We contrast results with earlier work in Section 6. Directions for further work in are listed in Section 7.

To save space, proofs of lemmas are omitted from this paper. For a discussion that includes a proof for every lemma, we refer to the extended version of this paper [8].

2 Preliminaries

For S a set, we write 2^S for the set of all subsets of S, and $\binom{S}{2}$ for the set of *multisets* over S of size two. An element of $\binom{S}{2}$ containing $s_1, s_2 \in S$ is written $\{\!|s_1, s_2|\!\}$; note that $\{\!|s_1, s_2|\!\} = \{\!|s_2, s_1|\!\}$. We use the symbols ϕ and ψ to denote multisets. If S and I are sets, and for every $i \in I$ there exists an $s_i \in S$, we call $(s_i)_{i \in I}$ an *I-indexed family over S*. We sometimes write $(s_i)_{i \in I} \in S$ to signify that the latter is a family over S. We say that a relation $\prec\, \subseteq S \times S$ is a *strict order* on S if it is irreflexive, asymmetric and transitive. We refer to $\prec$ as *well-founded* if there are no infinite descending $\prec$-chains, i.e., no family $(s_n)_{n \in \mathbb{N}} \in S$ such that $\forall n \in \mathbb{N}, s_{n+1} \prec s_n$. Throughout the paper we fix a finite set Σ called the *alphabet*, whose elements are symbols usually denoted with a and b. Lastly, if $\rightarrow\, \subseteq X \times Y \times Z$ is a ternary relation, we write $x \xrightarrow{y} z$ instead of $\langle x, y, z \rangle \in\, \rightarrow$.

2.1 Pomsets

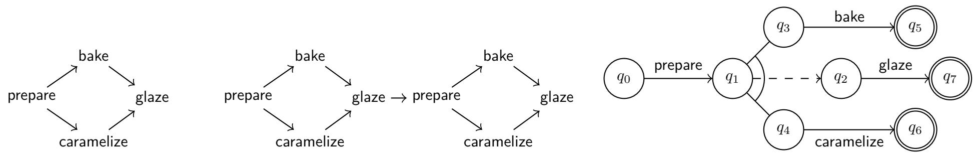
Partially-ordered multiset, or *pomset* [3] for short, generalise words to a setting where events (elements from Σ) may take place not just sequentially, but also in parallel.

▶ **Definition 2.1.** A *labelled poset* is a tuple $\langle U, \leq_U, \lambda_U \rangle$ consisting of a *carrier* set U, a partial order $\leq_U$ on U and a *labelling* function $\lambda_U : U \to \Sigma$. A *labelled poset isomorphism* is a bijection between poset carriers that bijectively preserves the labels and the ordering. A *pomset* is an isomorphism class of labelled posets; in practice, it is a labelled poset up-to bijective renaming of elements in U. We write 1 for the empty pomset, Pom_Σ for the set of all pomsets and Pom_Σ^+ for the set of all the non-empty pomsets.

For instance, suppose a recipe for caramel-glazed cookies tells us to (i) *prepare* cookie dough (ii) *bake* cookies in the oven (iii) *caramelize* sugar (iv) *glaze* the finished cookies. Here, step (i) precedes steps (ii) and (iii). Furthermore, step (iv) succeeds both steps (ii) and (iii). A pomset representing this process could be $\langle C, \leq_C, \lambda_C \rangle$, where $C = \{(\text{i}), (\text{ii}), (\text{iii}), (\text{iv})\}$ and $\leq_C$ is such that (i) $\leq_C$ (ii) $\leq_C$ (iv) and (i) $\leq_C$ (iii) $\leq_C$ (iv); λ_C is as in the recipe.

Note that words are just finite pomsets with a total order. We will sometimes refer to $a \in \Sigma$ as the pomset with a single point labelled a (and the obvious order); such a pomset is called *primitive*. A pomset can be represented as a Hasse diagram, where nodes have labels in Σ. For instance, the Hasse diagram for the pomset C above is drawn in Figure 1a.

To simplify notation, we refer to a pomset by the carrier U of a labelled poset $\langle U, \leq_U, \lambda_U \rangle$ in this isomorphism class. We use the symbols U, V, W and X to denote pomsets. Pomsets being isomorphism classes, the content of the carrier of the chosen representative is of very



(a) Diagram for C. **(b)** Diagram for $C \cdot C$. **(c)** PA accepting C.

Figure 1 Hasse diagrams for pomsets and a pomset automaton accepting one.

little importance; it is the order and labelling that matters. For this reason, we tacitly assume that whenever we have two pomsets, we pick representatives that have disjoint carrier sets.

▶ **Definition 2.2.** The *width* of a pomset U, denoted $\|U\|$, is the size of the largest antichain in U with respect to $\leq_U$, i.e., the maximum $n \in \mathbb{N}$ such that there exist $u_1, u_2, \dots, u_n \in U$ that are not related by $\leq_U$.

The pomsets we work with in this paper have a finite carrier. As a result, $\|U\|$ is always defined. For instance, the width of the pomset C above is 2, because the nodes (ii) and (iii) are an antichain of size 2, and there is no antichain of size 3.

▶ **Definition 2.3.** Let U and V be pomsets. The *sequential composition* of U and V, denoted $U \cdot V$, is the pomset $\langle U \cup V, \leq_U \cup \leq_V \cup (U \times V), \lambda_U \cup \lambda_V \rangle$. The *parallel composition* of U and V, denoted $U \parallel V$, is the pomset $\langle U \cup V, \leq_U \cup \leq_V, \lambda_U \cup \lambda_V \rangle$. Here, $\lambda_U \cup \lambda_V$ is the function from $U \cup V$ to Σ that agrees with λ_U on U, and with λ_V on V.

Note that 1 is the unit for both sequential and parallel composition. Sequential composition forces the events in the left pomset to be ordered before those in the right pomset. An example, describing the pomset $C \cdot C$, is depicted in Figure 1b.

▶ **Definition 2.4.** The set of *series-rational* pomsets is the smallest set that includes the primitive pomsets and is closed under sequential and parallel composition.

In this paper we will be mostly concerned with series-rational pomsets. For inductive reasoning about them, it is useful to record the following lemma.

▶ **Lemma 2.5** (Gischer [3]). *Let U be a series-rational pomset. Then, either (i) $U = a$ for some $a \in \Sigma$, or (ii) $U = V \cdot W$ for series-rational $V, W \in \mathsf{Pom}_\Sigma^+$, both strictly smaller than U, or (iii) $U = V \parallel W$ for series-rational $V, W \in \mathsf{Pom}_\Sigma^+$, both strictly smaller than U.*

2.2 Pomset languages

If a sequential program can exhibit multiple traces, we can group the words that represent these traces into a set called a *language*. By analogy, we can group the pomsets that represent the traces that arise from a parallel program into a set, which we refer to as a *pomset language*. Pomset languages are denoted by the symbols $\mathcal{U}$ and $\mathcal{V}$.

For instance, suppose that the recipe for glazed cookies may has an optional fifth step where chocolate sprinkles are spread over the cookies. In that case, there are *two* pomsets that describe a trace arising from the recipe, C^+ and C^-, either with or without the chocolate sprinkles. The pomset language $\mathcal{C} = \{C^-, C^+\}$ describes the new recipe.

▶ **Definition 2.6.** Let $\mathcal{U}$ be a pomset language. $\mathcal{U}$ has *bounded width* if there is $n \in \mathbb{N}$ such that for all $U \in \mathcal{U}$ we have $\|U\| \leq n$. The minimal such n is the *width* of $\mathcal{U}$, written $\|\mathcal{U}\|$.

The pomset languages considered in this paper have bounded width, and hence $\|\mathcal{U}\|$ is always defined. For instance, the width of $\mathcal{C}$ is 2, because the width of both C^+ and C^- is 2.

The sequential and parallel compositions of pomsets can be lifted to pomset languages. We also define a Kleene closure operator, similar to the one defined on languages of words.

▶ **Definition 2.7.** Let $\mathcal{U}$ and $\mathcal{V}$ be pomset languages. We define:

$$\mathcal{U} \cdot \mathcal{V} = \{U \cdot V : U \in \mathcal{U}, V \in \mathcal{V}\} \qquad \mathcal{U} \parallel \mathcal{V} = \{U \parallel V : U \in \mathcal{U}, V \in \mathcal{V}\} \qquad \mathcal{U}^* = \bigcup_{n \in \mathbb{N}} \mathcal{U}^n$$

Where $\mathcal{U}^0 = \{1\}$, and $\mathcal{U}^{n+1} = \mathcal{U} \cdot \mathcal{U}^n$ for all $n \in \mathbb{N}$.

Kleene closure models indefinite repetition. For instance, if our cookie recipe has a final step "repeat until enough cookies have been made", the pomset language $\mathcal{C}^*$ represents all possible traces of repetitions of the recipe; e.g., $C^+ \cdot C^+ \cdot C^- \in \mathcal{C}^*$ is the trace where first two batches of sprinkled cookies are made, followed by one without sprinkles.

2.3 Series-rational expressions

Just like a rational expression can be used to describe a regular structure of sequential events, a series-rational expression can be used to describe a regular structure of possibly parallel events. Series-rational expressions are rational expressions with parallel composition.

▶ **Definition 2.8.** The *series-rational expressions*, denoted $\mathcal{T}_\Sigma$, are formed by the grammar

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

We use the symbols d, e, f, g and h to denote series-rational expressions.

The semantics of a series-rational expression is given by a pomset language.

▶ **Definition 2.9.** The function $[\![-]\!] : \mathcal{T}_\Sigma \to 2^{\mathsf{Pom}_\Sigma}$ is defined inductively, as follows:

$$[\![0]\!] = \emptyset \qquad [\![a]\!] = \{a\} \qquad [\![1]\!] = \{1\} \qquad [\![e^*]\!] = [\![e]\!]^*$$

$$[\![e + f]\!] = [\![e]\!] \cup [\![f]\!] \qquad [\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!] \qquad [\![e \parallel f]\!] = [\![e]\!] \parallel [\![f]\!]$$

If $\mathcal{U} \in 2^{\mathsf{Pom}_\Sigma}$ such that $\mathcal{U} = [\![e]\!]$ for some $e \in \mathcal{T}_\Sigma$, then $\mathcal{U}$ is a *series-rational language*.

To illustrate, consider the pomset language $\mathcal{C}^* = \{C^+, C^-\}^*$, which describes the possible traces arising from indefinitely repeating the cookie recipe, optionally adding chocolate sprinkles at every repetition. We can describe the pomset language $\{C^-\}$ with the series-rational expression $c^- = \mathsf{prepare} \cdot (\mathsf{bake} \parallel \mathsf{caramelize}) \cdot \mathsf{glaze}$, and $\{C^+\}$ by $c^+ = c^- \cdot \mathsf{sprinkle}$, which yields the series-rational expression $c = c^- + c^+$ for $\mathcal{C}$. By construction, $[\![c^*]\!] = \mathcal{C}^*$.

2.4 Additive congruence

From the definition of $[\![-]\!]$, we can already see that series-rational expressions that are equal modulo associativity, commutativity and idempotence of the $+$-operator, as well as the unit 0, are also semantically equivalent. We furthermore identify series-rational expressions with an empty language. We formalize this identification by defining an congruence relation on series-rational expressions that will be useful throughout the paper.

▶ **Definition 2.10.** We define $\simeq$ as the smallest congruence on $\mathcal{T}_\Sigma$ such that:

$$e_1 + 0 \simeq e_1 \qquad e_1 + e_1 \simeq e_1 \qquad e_1 + e_2 \simeq e_2 + e_1 \qquad e_1 + (e_2 + e_3) \simeq (e_1 + e_2) + e_3$$

$$0 \cdot e_1 \simeq 0 \qquad e_1 \cdot 0 \simeq 0 \qquad 0 \parallel e \simeq 0 \qquad e \parallel 0 \simeq 0$$

When $\{\!|g, h|\!\}, \{\!|g', h'|\!\} \in \binom{\mathcal{T}_\Sigma}{2}$ such that $g \simeq g'$ and $h \simeq h'$, we write $\{\!|g, h|\!\} \simeq \{\!|g', h'|\!\}$.

Thus, when we claim that $e \simeq e'$, we say that e is equal to e', modulo the removal or addition of 0-terms, the merging of identical terms and the reordering or re-bracketing of terms (possibly within subterms). Moreover, congruence is sound with respect to the semantics, and it identifies all expressions that have an empty denotational semantics.

▶ **Lemma 2.11.** *Let $e, f \in \mathcal{T}_\Sigma$. If $e \simeq f$, then $[\![e]\!] = [\![f]\!]$. Also, $e \simeq 0$ if and only if $[\![e]\!] = \emptyset$.*

There is a simple linear time decision procedure to test whether two expressions are congruent. This justifies our using this relation to build finite automata later on. As a by-product, we get that the emptiness problem for series-rational expressions is linear time decidable.

3 Pomset Automata

We are now ready to describe an automaton model that recognises series-rational languages.

▶ **Definition 3.1.** A *pomset automaton (PA)* is a tuple $\langle Q, \delta, \gamma, F \rangle$ where Q is a set of *states*, with $F \subseteq Q$ the *accepting states*, $\delta : Q \times \Sigma \to Q$ is a function called the *sequential transition function*, $\gamma : Q \times \binom{Q}{2} \to Q$ is a function called the *parallel transition function*.

Note that we do not fix an initial state. As a result, a PA does not define a single pomset language but rather a mapping from its states to pomset languages. The language of a state is defined in terms of a trace relation that involves the transitions of both δ and γ. For the remainder of this section, we fix a PA $A = \langle Q, \delta, \gamma, F \rangle$, and a state $q \in Q$. To simplify matters later on, we assume that A has a state $\bot \in Q - F$ such that, for every $a \in \Sigma$, it holds that $\delta(\bot, a) = \bot$ and, for every $\phi \in \binom{Q}{2}$, it holds that $\gamma(\bot, \phi) = \bot$. Such a *sink state* is particularly useful for γ: for a fixed $q \in Q$ not all $\phi \in \binom{Q}{2}$ may give a value of $\gamma(q, \phi)$ that contributes to the language accepted by q. These inputs can be mapped to $\bot$ by γ. Equivalently, we can allow γ to be a partial function; we chose γ as a total function so as not to clutter the definition of derivatives in Section 4.

▶ **Definition 3.2.** $\rightarrow_A \subseteq Q \times \mathsf{Pom}_\Sigma^+ \times Q$ is the smallest relation[1] satisfying the rules

$$\frac{}{q \xrightarrow{a}_A \delta(q, a)} \qquad \frac{q \xrightarrow{U}_A q'' \qquad q'' \xrightarrow{V}_A q'}{q \xrightarrow{U \cdot V}_A q'} \qquad \frac{r \xrightarrow{U}_A r' \in F \qquad s \xrightarrow{V}_A s' \in F}{q \xrightarrow{U \parallel V}_A \gamma(q, \{\!|r, s|\!\})}$$

We also define $\twoheadrightarrow_A \subseteq Q \times \mathsf{Pom}_\Sigma \times Q$ by $q \overset{U}{\twoheadrightarrow}_A q'$ if and only if $q' = q$ and $U = 1$, or $q \xrightarrow{U}_A q'$. The *language* of A at $q \in Q$, denoted $L_A(q)$, is the set $\{U : \exists q' \in F.\ q \overset{U}{\twoheadrightarrow}_A q'\}$. We say that A *accepts* the language $\mathcal{U}$ if there exists a $q \in Q$ such that $L_A(q) = \mathcal{U}$.

Intuitively, γ ensures that when a process forks at state q into subprocesses starting at r and s, if each of those reaches an accepting state, then the processes can join at $\gamma(q, \{\!|r, s|\!\})$.

[1] The relation $\rightarrow_A$ should not be thought of as deterministic; for fixed $q \in Q$ and $U \in \mathsf{Pom}_\Sigma^+$, there may be multiple distinct $q' \in Q$ such that $q \xrightarrow{U}_A q'$— see the extended version [8] for additional information.

We purposefully omit the empty pomset 1 as a label in $\rightarrow_A$; doing so would open up the possibility of having traces of the form $q \xrightarrow{1}_A q'$ with $q \neq q'$ (i.e., "silent transitions" or "ϵ-transitions") for example by defining $\gamma(q, \{\!|r, s|\!\}) = q'$ for some $r, s \in F$. Avoiding transitions of this kind allows us to prove claims about $\rightarrow_A$ by induction on the pomset size, and leverage Lemma 2.5 in the process to disambiguate between the rules that apply. By extension, we can prove claims about $\twoheadrightarrow_A$ and L_A by treating $U = 1$ as a special case.

We draw a PA in a way similar to finite automata: each state (except $\perp$) is a vertex, and accepting states are marked by a double border. To represent sequential transitions, we draw labelled edges; for instance, in Figure 1c, $\delta(q_0, \mathsf{prepare}) = q_1$. To represent parallel transitions, we draw hyper-edges; for instance, in Figure 1c, $\gamma(q_1, \{\!|q_3, q_4|\!\}) = q_2$. To avoid clutter, we do not draw either of these edges types the target state is $\perp$. It is not hard to verify that the pomset C of the earlier example is accepted by the PA in Figure 1c.

In principle, the state space of a PA can be infinite; we use this later on to define a PA that has all possible series-rational expressions as states. It is however also useful to know when we can prune an infinite PA into a finite PA while preserving some accepted languages.

Note that it is not sufficient to talk about reachable states, i.e., states that appear in the target of some trace; we must also include states that are "meaningful" starting points for subprocesses. To do this, we first need a handle on these starting points. Specifically, we are interested in the states where (1) joining and forking yields a state that contributes to the behaviour of the PA, and (2) these states may join again, because they are not the sink state. This is captured in the definition below.

▶ **Definition 3.3.** The *support* of q, written $\pi_A(q)$, is the largest subset of $\binom{Q}{2}$ such that if $\{\!|r, s|\!\} \in \pi_A(q)$, then $\gamma(q, \{\!|r, s|\!\}) \neq \perp$, and $r, s \neq \perp$.

We can now talk about subsets of states of an automaton that are closed, in the sense that the relevant part of a transition function has input and output confined to this set. As a result, we can confine the structure of a given PA to a closed set.

▶ **Definition 3.4.** A set of states $Q' \subseteq Q$ is *closed* when the following rules are satisfied

$$\frac{}{\perp \in Q'} \qquad \frac{q \in Q' \qquad a \in \Sigma}{\delta(q, a) \in Q'} \qquad \frac{q \in Q' \qquad \phi \in \pi_A(q)}{\gamma(q, \phi) \in Q'} \qquad \frac{q \in Q' \qquad \{\!|r, s|\!\} \in \pi_A(q)}{r, s \in Q'}$$

If Q' is closed, the *generated sub-PA* of A induced by Q', denoted $A\!\restriction_{Q'}$, is the tuple $\langle Q', \delta\!\restriction_{Q'}, \gamma\!\restriction_{Q'}, Q' \cap F\rangle$ where $\delta\!\restriction_{Q'}$ and $\gamma\!\restriction_{Q'}$ are the restrictions of δ and γ to Q'.

Because the relevant parts of the transition functions are preserved, it is not altogether surprising that languages of a generated sub-PA are also languages of the original PA.

▶ **Lemma 3.5.** *Let $Q' \subseteq Q$ be closed. If $q \in Q'$, then $L_{A\restriction_{Q'}}(q) = L_A(q)$.*

We now work out how to find a closed subset of states that contains a particular state. The first step is to characterize the states reachable from q by means of a trace.

▶ **Definition 3.6.** The *reach* of q, written $\rho_A(q)$, is the smallest set satisfying the rules

$$\frac{}{q, \perp \in \rho_A(q)} \qquad \frac{q' \in \rho_A(q) \qquad a \in \Sigma}{\delta(q', a) \in \rho_A(q)} \qquad \frac{q' \in \rho_A(q) \qquad \phi \in \pi_A(q)}{\gamma(q', \phi) \in \rho_A(q)}$$

The reach of a state is closely connected to the states that can be reached from q through the trace relation of the automaton, in the following way:

▶ **Lemma 3.7.** *The reach of q contains $\{q' \in Q : \exists U \in \mathsf{Pom}_\Sigma^+.\ q \xrightarrow{U}_A q'\} \cup \{\bot, q\}$. Moreover, if $\bot$ is the only state of A whose language is empty, this containment is an equality.*

Note that $\rho_A(q)$ is not necessarily closed: we also need the states required by the fourth rule of closure in Definition 3.4. Thus, if we want to "close" $\rho_A(q)$ by adding the support of its contents, we need to find finite, closed sets of states that contain branching points. In order to do this inductively, we propose the following subclass of PAs.

▶ **Definition 3.8.** We say that A is *fork-acyclic* if there exists a *fork hierarchy*, which is a strict order $\prec_A \subseteq Q \times Q$ such that the following rules are satisfied.

$$\frac{\{\!\!\{r, s\}\!\!\} \in \pi_A(q)}{r, s \prec_A q} \qquad \frac{a \in \Sigma \qquad r \prec_A \delta(q, a)}{r \prec_A q} \qquad \frac{\phi \in \pi_A(q) \qquad r \prec_A \gamma(q, \phi)}{r \prec_A q}$$

The fork hierarchy is connected with the reach of a state in the following way.

▶ **Lemma 3.9.** *Let $q', r \in Q$. If A is fork-acyclic, $q' \in \rho_A(q)$ and $r \prec_A q'$, then $r \prec_A q$.*

The term *fork-acyclic* has been used in literature for similar automata [15, 6]. However, in op. cit., it is defined in terms of the traces that arise from the transition structure of the automaton. In contrast, our definition is purely syntactic: it imposes an order on states such that a forks cannot be nested. To show that, as in [15], our definition similarly implies languages of the PA have bounded width, we present the following lemma. Since the state space of a PA can be infinite, we additionally require that the fork hierarchy is well-founded.

▶ **Lemma 3.10.** *If A is fork-acyclic and $\prec_A$ is well-founded then $L_A(q)$ is of finite width.*

We introduce the notion of a *bounded PA*, which is sufficient to guarantee the existence of a closed, finite subset containing a given state, even when the PA has infinitely many states.

▶ **Definition 3.11.** If A is fork-acyclic, we say that it is *bounded* if $\prec_A$ is well-founded, and for all $q \in Q$, both $\pi_A(q)$ and $\rho_A(q)$ are finite.

▶ **Theorem 3.12.** *If A is bounded, there exists a finite set of states $Q_q \subseteq Q$ that is closed and contains q.*

Proof. The proof proceeds by $\prec_A$-induction; this is sound, because $\prec_A$ is well-founded.

Suppose the claim holds for all $r \in Q$ with $r \prec_A q$. If $q' \in \rho_A(q)$ and $\{\!\!\{r, s\}\!\!\} \in \pi_A(q')$, then $r \prec_A q'$ and thus $r \prec_A q$ by Lemma 3.9; by induction we obtain for every such r a finite set of states $Q_r \subseteq Q$ that is closed in A and contains r. We choose:

$$Q_q = \rho_A(q) \cup \bigcup \{Q_r : q' \in \rho_A(q),\ \{\!\!\{r, s\}\!\!\} \in \pi_A(q')\}$$

This set is finite because $\rho_A(q)$ and $\pi_A(q')$ are finite for all $q, q' \in Q$ since A is bounded. To see that Q_q is closed in A, it suffices to show that the last rule of closure holds for $q' \in \rho_A(q)$; it does, since if $q' \in \rho_A(q)$ and $\{\!\!\{r, s\}\!\!\} \in \pi_A(q')$, then $r \in Q_r$ and $s \in Q_s$, thus $r, s \in Q_q$. ◀

4 Expressions to automata

We now turn our attention to the task of translating a series-rational expression e into a PA that accepts $[\![e]\!]$. We employ Brzozowski's method [2] to construct a single *syntactic PA* where every series-rational expression is a state accepting exactly its denotational semantics. To this end we must define which expressions are accepting, and how the sequential and parallel transition functions (*derivatives*, in Brzozowski's vocabulary) transform states.

We start with the accepting states. In Brzozowski's original construction, a state is accepting if the denotational semantics of its expression includes the empty word. Analogously, our accepting terms are the terms whose denotational semantics includes the empty pomset.

▶ **Definition 4.1.** We define the set F_Σ to be the smallest subset of $\mathcal{T}_\Sigma$ satisfying the rules:

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

It is not hard to see that $e \in F_\Sigma$ if and only if $1 \in [\![e]\!]$. We use $e \star f$ as a shorthand for f if $e \in F_\Sigma$, and $0 \in \mathcal{T}_\Sigma$ otherwise. Let $\mathcal{E}$ be an equation, we write $[\mathcal{E}]$ as a shorthand for $1 \in \mathcal{T}_\Sigma$ if $\mathcal{E}$ holds, and $0 \in \mathcal{T}_\Sigma$ otherwise. We now define sequential and parallel derivatives:

▶ **Definition 4.2.** We define the function $\delta_\Sigma : \mathcal{T}_\Sigma \times \Sigma \to \mathcal{T}_\Sigma$ as follows:

$$\delta_\Sigma(0, a) = 0 \qquad \delta_\Sigma(1, a) = 0 \qquad \delta_\Sigma(b, a) = [a = b] \qquad \delta_\Sigma(e^*, a) = \delta_\Sigma(e, a) \cdot e^*$$

$$\delta_\Sigma(e + f, a) = \delta_\Sigma(e, a) + \delta_\Sigma(f, a) \qquad \delta_\Sigma(e \cdot f, a) = \delta_\Sigma(e, a) \cdot f + e \star \delta_\Sigma(f, a)$$

$$\delta_\Sigma(e \parallel f, a) = e \star \delta_\Sigma(f, a) + f \star \delta_\Sigma(e, a)$$

Furthermore, the function $\gamma_\Sigma : \mathcal{T}_\Sigma \times \binom{\mathcal{T}_\Sigma}{2} \to \mathcal{T}_\Sigma$ is defined as follows:

$$\gamma_\Sigma(0, \phi) = 0 \qquad \gamma_\Sigma(1, \phi) = 0 \qquad \gamma_\Sigma(b, \phi) = 0 \qquad \gamma_\Sigma(e^*, \phi) = \gamma_\Sigma(e, \phi) \cdot e^*$$

$$\gamma_\Sigma(e + f, \phi) = \gamma_\Sigma(e, \phi) + \gamma_\Sigma(f, \phi) \qquad \gamma_\Sigma(e \cdot f, \phi) = \gamma_\Sigma(e, \phi) \cdot f + e \star \gamma_\Sigma(f, \phi)$$

$$\gamma_\Sigma(e \parallel f, \phi) = [\phi \simeq \{\!|e, f|\!\}] + e \star \gamma_\Sigma(f, \phi) + f \star \gamma_\Sigma(e, \phi)$$

The definition of δ_Σ coincides with Brzozowski's derivative on rational expressions. The definition of γ_Σ mimics the definition of δ_Σ on non-parallel terms except $b \in \Sigma$.

The definition of γ_Σ on parallel terms includes (in the first term) the possibility that the starting states provided to the parallel transition function are (congruent to) the operands of the parallel, in which case the target join state is the accepting state 1. The other two terms (as well as the definition of δ_Σ on a parallel term) account for the fact that if $1 \in [\![e]\!]$, then $[\![f]\!] \subseteq [\![e \parallel f]\!]$. Since we do not allow traces labelled with the empty pomset, traces that originate from these operands are thus lifted to the composition when necessary.

▶ **Definition 4.3.** The *syntactic PA* is the PA $A_\Sigma = \langle \mathcal{T}_\Sigma, \delta_\Sigma, \gamma_\Sigma, F_\Sigma \rangle$.

We use L_Σ as a shorthand for L_{A_Σ}, and $\rightarrow_\Sigma$ ($\twoheadrightarrow_\Sigma$) as a shorthand for $\rightarrow_{A_\Sigma}$ ($\twoheadrightarrow_{A_\Sigma}$).

The remainder of this section is devoted to showing that if $e \in \mathcal{T}_\Sigma$, then $L_\Sigma(e) = [\![e]\!]$.

4.1 Traces of congruent states

In the analysis of the syntactic trace relation $\rightarrow_\Sigma$, we often encounter sums of terms. To work with these terms, it is often useful to identify sums that are the same modulo associativity, commutativity, idempotence and 0-unit of $+$. In this section, we establish that such an identification is in fact sound, in the sense that if two expressions are related by $\simeq$, then the languages accepted by the states representing those expressions are also identical.

In the first step towards this goal, we show that F_Σ is well-defined with respect to $\simeq$.

▶ **Lemma 4.4.** *Let $e, f \in \mathcal{T}_\Sigma$ be such that $e \simeq f$. Then $e \in F_\Sigma$ if and only if $f \in F_\Sigma$.*

Also, δ_Σ and γ_Σ are well-defined with respect to $\simeq$, in the following sense:

▶ **Lemma 4.5.** *Let $e, f \in \mathcal{T}_\Sigma$ such that $e \simeq f$. If $a \in \Sigma$, then $\delta_\Sigma(e, a) \simeq \delta_\Sigma(f, a)$. Moreover, if $\phi = \{\!\{g, h\}\!\} \in \binom{\mathcal{T}_\Sigma}{2}$ with $g, h \not\simeq 0$, then $\gamma_\Sigma(e, \phi) \simeq \gamma_\Sigma(f, \phi)$, and if $\psi \in \binom{\mathcal{T}_\Sigma}{2}$ with $\phi \simeq \psi$, then $\gamma_\Sigma(e, \phi) = \gamma_\Sigma(e, \psi)$.*

With these lemmas in hand, we can show that $\simeq$ is a "bisimulation" with respect to $\rightarrow_\Sigma$.

▶ **Lemma 4.6.** *Let $e, f \in \mathcal{T}_\Sigma$ be such that $e \simeq f$. If $e \xrightarrow{U}_\Sigma e'$, then there exists an $f' \in \mathcal{T}_\Sigma$ such that $f \xrightarrow{U}_\Sigma f'$ and $e' \simeq f'$.*

Let I be a finite set, and let $(e_i)_{i \in I}$ be an I-indexed family of terms. In the sequel, we sometimes abuse notation and treat $\sum_{i \in I} e_i$ as a term, where the e_i are summed in some arbitrary order or bracketing. The lemmas above guarantee that the precise choice of representing this sum as a term makes no matter with regard to the traces allowed.

4.2 Trace deconstruction

We proceed with a series of lemmas that characterise reachable states in the syntactic PA. More precisely, we show that expressions of reachable states can be written as sums, each term of which is related to a trace of a subexpression of the starting state. For this reason, we refer to these observations as *trace deconstruction lemmas*: they deconstruct a trace into traces of "smaller" expressions. The purpose of these lemmas is twofold; in Section 4.4, they are used to characterise the languages of expressions as they appear in the syntactic PA, while in Section 4.5 they allow us to bound the reach of an expression.

We start by analysing the traces that originate in base terms, such as 0, 1, or $a \in \Sigma$.

▶ **Lemma 4.7.** *Let $e, e' \in \mathcal{T}_\Sigma$ and $U \in \mathsf{Pom}^+_\Sigma$ such that $e \xrightarrow{U}_\Sigma e'$. If $e \in \{0, 1\}$, then $e' = 0$. Furthermore, if $e = b \in \Sigma$, then either $e' = 1$ and $U = b$, or $e' = 0$.*

Note, however, that the first claim above does not imply that 0 and 1 are indistinguishable as far as traces are concerned, for $0 \notin F_\Sigma$ while $1 \in F_\Sigma$.

We also consider the traces that originate in a sum of terms. The intuition here is that the input is processed by both terms simultaneously, and thus the target state must be the sum of the states that are the result of processing the input for to each term individually.

▶ **Lemma 4.8.** *Let $e_1, e_2 \in \mathcal{T}_\Sigma$ and $U \in \mathsf{Pom}^+_\Sigma$. If $e_1 + e_2 \xrightarrow{U}_\Sigma e'$, then there exist $e'_1, e'_2 \in \mathcal{T}_\Sigma$ such that $e' = e'_1 + e'_2$, and $e_1 \xrightarrow{U}_\Sigma e'_1$ and $e_2 \xrightarrow{U}_\Sigma e'_2$.*

We now consider the traces starting in a state that is a sequential composition. The intuition here is that the syntactic PA must first proceed through the left operand, before it can proceed to process the right operand. Thus, either the pomset is processed by the left operand entirely, or we should be able to split the pomset in two sequential parts: the first part is processed by the left operand, and the second by the right operand.

▶ **Lemma 4.9.** *Let $e_1, e_2 \in \mathcal{T}_\Sigma$ and $U \in \mathsf{Pom}^+_\Sigma$ be such that $e_1 \cdot e_2 \xrightarrow{U}_\Sigma f$. There exist an $f' \in \mathcal{T}_\Sigma$ and a finite set I, as well as I-indexed families of terms $(f'_i)_{i \in I} \in F_\Sigma$ and $(f_i)_{i \in I} \in \mathcal{T}_\Sigma$ and of pomsets $(U'_i)_{i \in I} \in \mathsf{Pom}_\Sigma$ and $(U_i)_{i \in I} \in \mathsf{Pom}^+_\Sigma$, such that:*

- *$f \simeq f' \cdot e_2 + \sum_{i \in I} f_i$ and $e_1 \xrightarrow{U}_\Sigma f'$*
- *for all $i \in I$, $e_1 \overset{U'_i}{\twoheadrightarrow}_\Sigma f'_i$, $e_2 \xrightarrow{U_i}_\Sigma f_i$, and $U = U'_i \cdot U_i$.*

The next deconstruction lemma concerns traces originating in a parallel composition. Intuitively, the syntactic PA either processes parallel components of the pomset, or processes according to one operand, provided that the other operand allows immediate acceptance.

▶ **Lemma 4.10.** *If $e_1 \parallel e_2 \xrightarrow{U}_\Sigma f$, then there exist $f_1, f_2, f_3 \in \mathcal{T}_\Sigma$, such that*

- $f \simeq f_1 + f_2 + f_3$
- *either $f_1 = 0$, or $e_2 \in F_\Sigma$ and $e_1 \xrightarrow{U}_\Sigma f_1$*
- *either $f_2 = 0$, or $e_1 \in F_\Sigma$ and $e_2 \xrightarrow{U}_\Sigma f_2$*
- *either $f_3 = 0$, or $f_3 = 1$ and there exist $e_1', e_2' \in \mathcal{T}_\Sigma$ and $f_1', f_2' \in F_\Sigma$ and $U_1, U_2 \in \mathsf{Pom}_\Sigma^+$ such that $e_1 \simeq e_1'$ and $e_2 \simeq e_2'$ and $U = U_1 \parallel U_2$ and $e_1' \xrightarrow{U_1}_\Sigma f_1'$ and $e_2' \xrightarrow{U_2}_\Sigma f_2'$.*

Finally, we analyse the reachable states of an expression of the form e^*. The intuition here is that, starting in e^*, the PA can process traces originating in e indefinitely. The trace should thus be sequentially decomposable, with each component the label of a trace originating in e. Furthermore, all but the last target state of these traces should be accepting.

▶ **Lemma 4.11.** *If $e^* \xrightarrow{U}_\Sigma f$, then there exists a finite set I and an I-indexed family of finite sets $(J_i)_{i \in I}$, as well as I-indexed families of terms $(f_i)_{i \in I} \in \mathcal{T}_\Sigma$ and of pomsets $(U_i)_{i \in I} \in \mathsf{Pom}_\Sigma^+$, and for all $i \in I$ also J_i-indexed families of terms $(f_{i,j})_{j \in J_i} \in F_\Sigma$ and of pomsets $(U_{i,j})_{j \in J_i} \in \mathsf{Pom}_\Sigma^+$, such that $f \simeq \sum_{i \in I} f_i \cdot e^*$, and for all $i \in I$:*

- $e \xrightarrow{U_i}_\Sigma f_i$
- *for all $j \in J_i$ we have that $e \xrightarrow{U_{i,j}}_\Sigma f_{i,j}$*
- *$U = U_i' \cdot U_i$, where U_i' is some concatenation of all $U_{i,j}$ for all $j \in J_i$.*

4.3 Trace construction

In the above, we learned how to deconstruct traces in the syntactic PA. To verify that the syntactic PA indeed accepts any series-rational pomset language, we also need to show the reverse, that is, how to *construct* traces in the syntactic PA from smaller traces. In this context it is often useful to work with the preorder obtained from $\simeq$.

▶ **Definition 4.12.** The relation $\lesssim \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_\Sigma$ is defined by $e \lesssim f$ if and only if $e + f \simeq f$.

The intuition to $e \lesssim f$ is that e consists of one or more terms that also appear in f, up to $\simeq$.

In analogy to Lemma 4.6, we show that $\lesssim$ is a "simulation" with respect to traces.

▶ **Lemma 4.13.** *Let $e, e', f \in \mathcal{T}_\Sigma$ be such that $e \lesssim f$. If $e \xrightarrow{U}_\Sigma e'$, then there exists an $f' \in \mathcal{T}_\Sigma$ such that $f \xrightarrow{U}_\Sigma f'$ and $e' \lesssim f'$. Furthermore, if $e \in F_\Sigma$, then $f \in F_\Sigma$.*

The following lemma tells us that we can create a trace labelled with the concatenation of the labels of two smaller traces, and starting in the sequential composition of the original starting states, provided that the first trace ends in an accepting state. Furthermore, the target state of the newly constructed trace contains the target state of the second trace.

▶ **Lemma 4.14.** *Let $e_1, e_2, f_2 \in \mathcal{T}_\Sigma$ and $f_1 \in F_\Sigma$. If $U, V \in \mathsf{Pom}_\Sigma^+$ are such that $e_1 \xrightarrow{U}_\Sigma f_1$ and $e_2 \xrightarrow{V}_\Sigma f_2$, then there exists an $f \in \mathcal{T}_\Sigma$ such that $e_1 \cdot e_2 \xrightarrow{U \cdot V}_\Sigma f$ with $f_2 \lesssim f$.*

We can also construct traces that start in a parallel composition. One way is to construct traces that start in each operand and reach an accepting state; we obtain a trace in their parallel composition almost trivially. If one of the operands is accepting, we can also construct a single trace that starts in the other operand and obtain a trace with the same label starting in the parallel construction. In both cases, we describe the target of the new trace using $\lesssim$.

▶ **Lemma 4.15.** *Let $e_1, e_2 \in \mathcal{T}_\Sigma$. The following hold:*

- *If $f_1, f_2 \in F_\Sigma$ and $U, V \in \mathsf{Pom}_\Sigma^+$ are such that $e_1 \xrightarrow{U}_\Sigma f_1$ and $e_2 \xrightarrow{V}_\Sigma f_2$, then there exists an $f \in \mathcal{T}_\Sigma$ such that $e_1 \parallel e_2 \xrightarrow{U \parallel V}_\Sigma f$ with $1 \lesssim f$.*
- *If $e_2 \in F_\Sigma$ (respectively $e_1 \in F_\Sigma$), and $f' \in \mathcal{T}_\Sigma$ and $U \in \mathsf{Pom}_\Sigma^+$ are such that $e_1 \xrightarrow{U}_\Sigma f'$ (respectively $e_2 \xrightarrow{U}_\Sigma f'$), then there exists an $f \in \mathcal{T}_\Sigma$ such that $e_1 \parallel e_2 \xrightarrow{U}_\Sigma f$ with $f' \lesssim f$.*

Lastly, we present a trace construction lemma to obtain traces originating in expressions of the form e^*. The idea here is that, given a finite number of traces that originate in e, where all (but possibly one) have an accepting state as their target, we can construct a trace originating in e^*, with a concatenation of the labels of the input traces as its label.

▶ **Lemma 4.16.** *Let $e, f_1, f_2, \ldots, f_n \in \mathcal{T}_\Sigma$ (with $n > 0$) be such that $f_1, f_2, \ldots, f_{n-1} \in F_\Sigma$. Also, let $U, U_1, U_2, \ldots, U_n \in \mathsf{Pom}_\Sigma^+$ be such that $U = U_1 \cdot U_2 \cdots U_n$. If for all $i \leq n$ it holds that $e \xrightarrow{U_i}_\Sigma f_i$, then there exists an $f \in \mathcal{T}_\Sigma$ such that $e^* \xrightarrow{U}_\Sigma f$, with $f_n \cdot e^* \lesssim f$.*

4.4 Soundness for the syntactic PA

With trace deconstruction and construction lemmas in our toolbox, we are ready to show that the syntactic PA indeed captures series-rational languages.

First, note that L_Σ can be seen as a function from $\mathcal{T}_\Sigma$ to Pom_Σ, like $[\![-]\!]$. To establish equality between L_Σ and $[\![-]\!]$, we first show that L_Σ enjoys the same homomorphic equalities as those in the definition of the semantic map, i.e., that $L_\Sigma(e)$ can be expressed in terms of L_Σ applied to subexpressions of e. The proofs of the equalities below follow a similar pattern: for the inclusion from left to right we use trace deconstruction lemmas to obtain traces for the component expressions, while for the inclusion from right to left we use trace construction lemmas to build traces for the composed expressions given the traces of the component expressions. We treat the case for the empty pomset separately almost everywhere.

▶ **Lemma 4.17.** *Let $e_1, e_2 \in \mathcal{T}_\Sigma$, and $a \in \Sigma$. The following equalities hold:*

$$L_\Sigma(0) = \emptyset \qquad L_\Sigma(1) = \{1\} \qquad L_\Sigma(a) = \{a\} \qquad L_\Sigma(e_1 + e_2) = L_\Sigma(e_1) \cup L_\Sigma(e_2)$$

$$L_\Sigma(e_1 \cdot e_2) = L_\Sigma(e_1) \cdot L_\Sigma(e_2) \qquad L_\Sigma(e_1 \parallel e_2) = L_\Sigma(e_1) \parallel L_\Sigma(e_2) \qquad L_\Sigma(e_1^*) = L_\Sigma(e_1)^*$$

It is now easy to establish that the Brzozowski construction for the syntactic PA is sound with respect to the denotational semantics of series-rational expressions.

▶ **Theorem 4.18.** *Let $e \in \mathcal{T}_\Sigma$. Then $L_\Sigma(e) = [\![e]\!]$.*

Proof. The proof proceeds by induction on e. In the base, $e = 0$, $e = 1$ or $e = a$ for some $a \in \Sigma$. In all cases, $L_\Sigma(e) = [\![e]\!]$ by Lemma 4.17. For the inductive step, there are four cases to consider: either $e = e_1 + e_2$, $e = e_1 \cdot e_2$, $e = e_1 \parallel e_2$ or $e = e_1^*$. In all cases, the claim follows from the induction hypothesis and the definition of $[\![-]\!]$, combined with Lemma 4.17. ◀

4.5 Bounding the syntactic PA

Ideally, we would like to obtain a single PA with finitely many states that recognizes $[\![e]\!]$ for a given $e \in \mathcal{T}_\Sigma$. Unfortunately, the syntactic PA is not bounded, and thus Theorem 3.12 does not apply. For instance, the requirement that $\rho_\Sigma(e)$ be finite for $e \in \mathcal{T}_\Sigma$ fails; consider the family of distinct terms $(e_n)_{n\in\mathbb{N}}$ defined by $e_0 = 1 \cdot a^*$ and $e_{n+1} = 0 \cdot a^* + e_n$ for $n \in \mathbb{N}$; it is not hard to show that $e_n \in \rho_\Sigma(a^*)$ for $n \in \mathbb{N}$, and thus conclude that $\rho_\Sigma(a^*)$ is infinite. We remedy this problem by quotienting the state space of the syntactic PA by congruence.

In what follows, we write $[e]$ for the congruence class of $e \in \mathcal{T}_\Sigma$ modulo $\simeq$, i.e., the set of all $e' \in \mathcal{T}_\Sigma$ such that $e \simeq e'$. We furthermore write $\mathcal{Q}_\Sigma$ for the set of all congruence classes of expressions in $\mathcal{T}_\Sigma$. We now leverage Lemma 4.5 to define a transition structure on $\mathcal{Q}_\Sigma$.

▶ **Definition 4.19.** We define $\delta_\simeq : \mathcal{Q}_\Sigma \times \Sigma \to \mathcal{Q}_\Sigma$ and $\gamma_\simeq : \mathcal{Q}_\Sigma \times \binom{\mathcal{Q}_\Sigma}{2} \to \mathcal{Q}_\Sigma$ as

$$\delta_\simeq([e], a) = [\delta_\Sigma(e, a)] \qquad \gamma_\simeq([e], \{\![f], [g]\}\!\}) = \begin{cases} [0] & f \simeq 0 \text{ or } g \simeq 0 \\ [\gamma_\Sigma(e, \{\!\{g, h\}\!\})] & \text{otherwise} \end{cases}$$

Furthermore, the set $F_\simeq$ is defined to be $\{[e] : e \in F_\Sigma\}$. The *quotiented syntactic PA* is the PA $A_\simeq = \langle \mathcal{Q}_\Sigma, \delta_\simeq, \gamma_\simeq, F_\simeq \rangle$.

▶ **Lemma 4.20.** *$\delta_\simeq$ and $\gamma_\simeq$, as well as $F_\simeq$, are well-defined.*

Just like for the syntactic PA, we abbreviate subscripts, for example by writing $\to_\simeq$ rather than $\to_{A_\simeq}$, and $L_\simeq$ rather than $L_{A_\simeq}$. Of course, we also want the quotiented syntactic PA to accept the same languages as the syntactic PA. To that end, we show that the trace relations of the syntactic PA and the quotiented syntactic PA correspond.

▶ **Lemma 4.21.** *Let $e, f \in \mathcal{T}_\Sigma$ and $U \in \mathsf{Pom}_\Sigma^+$. If $e \xrightarrow{U}_\Sigma f$, then $[e] \xrightarrow{U}_\simeq [f]$. If $[e] \xrightarrow{U}_\simeq [f]$, then there exists an $f \in \mathcal{T}_\Sigma$ with $f \simeq f'$ and $e \xrightarrow{U}_\Sigma f'$.*

▶ **Theorem 4.22.** *Let $e \in \mathcal{T}_\Sigma$. Then $L_\Sigma(e) = L_\simeq([e])$.*

Proof. By definition, $e \in F_\Sigma$ if and only if $[e] \in F_\simeq$. It then follows that $1 \in L_\Sigma(e)$ if and only if $1 \in L_\simeq([e])$. The remaining cases are covered by Lemma 4.21. ◀

▶ **Corollary 4.23.** *The state $[0]$ is the only state in the quotiented syntactic PA with an empty language.*

To show that the quotiented syntactic PA is bounded, we construct a strict order $\prec$ on $\mathcal{Q}_\Sigma$ that satisfies the rules of Definition 3.8.

▶ **Definition 4.24.** Let $e \in \mathcal{T}_\Sigma$. The *parallel depth* of e, denoted $(\!|e|\!)$, is defined to be 0 when $e \simeq 0$, and otherwise:

$$(\!|1|\!) = 0 \qquad (\!|a|\!) = 1 \qquad (\!|e_1 + e_2|\!) = \max((\!|e_1|\!), (\!|e_2|\!))$$

$$(\!|e_1 \cdot e_2|\!) = \max((\!|e_1|\!), (\!|e_2|\!)) \qquad (\!|e_1 \parallel e_2|\!) = \max((\!|e_1|\!), (\!|e_2|\!)) + 1 \qquad (\!|e_1^*|\!) = (\!|e_1|\!)$$

It is easy to show that the parallel depth of an expression is also well-defined on the congruence classes of $\simeq$. This allows us to define the desired strict order on $\mathcal{Q}_\Sigma$.

▶ **Lemma 4.25.** *Let $e, e' \in \mathcal{T}_\Sigma$ be such that $e \simeq e'$. Then $(\!|e|\!) = (\!|e'|\!)$.*

▶ **Definition 4.26.** The relation $\prec \subseteq \mathcal{Q}_\Sigma \times \mathcal{Q}_\Sigma$ is such that $[e] \prec [f]$ if and only if $(\!|e|\!) < (\!|f|\!)$.

As it turns out, $\prec$ is the desired fork hierarchy.

▶ **Lemma 4.27.** *The quotiented syntactic PA is fork-acyclic, with fork hierarchy $\prec$.*

We now investigate the reach and support of a state in the quotiented syntactic PA, using the deconstruction lemmas showed in Section 4.2, as well as Lemma 4.21.

▶ **Lemma 4.28.** *Let $e \in \mathcal{T}_\Sigma$. Then $\rho_\simeq([e])$ and $\pi_\simeq([e])$ are finite.*

▶ **Theorem 4.29.** *The quotiented syntactic PA is bounded.*

Proof. By Lemma 4.27, we know that $A_\simeq$ is fork-acyclic. Furthermore, $\prec$ is well-founded by construction: every term has finite depth. By Lemma 4.28, we know that $\rho_\simeq([e])$ and $\pi_\simeq([e])$ are finite for all $[e] \in \mathcal{Q}_\Sigma$. All requirements are now validated. ◀

Since the quotiented syntactic PA is bounded, it restricts to a finite PA for any expression.

▶ **Corollary 4.30.** *Let $e \in \mathcal{T}_\Sigma$. There exists a finite PA A_e that accepts $[\![e]\!]$.*

5 Automata to expressions

To associate with every state q in a bounded PA $A = \langle Q, \delta, \gamma, F\rangle$ a series-rational expression e_q such that $[\![e_q]\!] = L_A(q)$, we modify the procedure for associating a regular expression with a state in a finite automaton described in [11]. The modification consists of adding parallel terms to the expression associated with q whenever a fork in q contributes to its language, i.e., whenever $\{\!|r, s|\!\} \in \pi_A(q)$.

In view of the special treatment of 1 in the semantics of PAs, it is convenient to first define expressions e_q^+ with the property that $[\![e_q^+]\!] = L_A(q) - \{1\}$; then we can define e_q by $e_q = e_q^+ + [q \in F]$. The definition of e_q^+ proceeds by induction on the well-founded partial order $\prec_A$ associated with a bounded PA. That is, when defining e_q^+ we assume the existence of expressions $e_{q'}^+$ for all $q' \in Q$ such that $q' \prec_A q$.

First, however, we shall define auxiliary expressions $e_{qq'}^{Q'}$ for suitable choices of $Q' \subseteq Q$ and of $q, q' \in Q$. Intuitively, $e_{qq'}^{Q'}$ denotes the pomset language characterizing all paths from q to q' with all intermediate states in Q'; e_q^+ can then be defined as the summation of all $e_{qq'}^{\rho_A(q)}$ with $q' \in F \cap \rho_A(q)$.

▶ **Definition 5.1.** Let Q' be a finite subset of Q, and assume that for all $r \in Q$ such that $r \prec_A q$ for some $q \in Q'$ there exists a series-rational expression $e_r^+ \in \mathcal{T}_\Sigma$ such that $[\![e_r^+]\!] = L_A(q) - \{1\}$. For all $Q'' \subseteq Q'$ and $q, q' \in Q'$, we define a series-rational expression $e_{qq'}^{Q''}$ by induction on the size of Q'', as follows:

1. If $Q'' = \emptyset$, then let $\widetilde{\Sigma}$ be the set $\{a \in \Sigma : q' = \delta(q, a)\}$, and let $\widetilde{Q}$ be the set $\{\phi \in \pi_A(q) : \gamma(q, \phi) = q'\}$. We define
$$e_{qq'}^{Q''} = \sum_{a \in \widetilde{\Sigma}} a + \sum_{\{\!|r,s|\!\} \in \widetilde{Q}} e_r^+ \parallel e_s^+ \ .$$
2. Otherwise, we choose a $q'' \in Q''$ and define
$$e_{qq'}^{Q''} = e_{qq'}^{Q''-\{q''\}} + e_{qq''}^{Q''-\{q''\}} \cdot (e_{q''q''}^{Q''-\{q''\}})^* \cdot e_{q''q'}^{Q''-\{q''\}} \ .$$

Note that e_r^+ and e_s^+, appearing in the first clause of the definition of $e_{qq'}^{Q''}$, exist by assumption, for by fork-acyclicity we have that $r, s \prec_A q \in Q'$.

▶ **Theorem 5.2.** *Let Q' be a finite subset of Q and assume that for all $r \in Q$ such that $r \prec_A q$ for some $q \in Q'$ there exists a series-rational expression $e_r^+ \in \mathcal{T}_\Sigma$ with $[\![e_r^+]\!] = L_A(q) - \{1\}$. For all $q, q' \in Q'$, for all $Q'' \subseteq Q'$, and for all $U \in \mathsf{Pom}_\Sigma^+$, we have that $q \xrightarrow{U}_A q'$ according to some path that only visits states in Q'' if, and only if, $U \in [\![e_{qq'}^{Q''}]\!]$.*

Using the auxiliary series-rational expressions $e_{qq'}^{Q''}$, we can now associate series-rational expressions $e_q, e_q^+ \in \mathcal{T}_\Sigma$ with every $q \in Q$, defining e_q^+ by $e_q^+ = \sum_{q' \in \rho_A(q) \cap F} e_{qq'}^{\rho_A(q)}$ and $e_q = e_q^+ + [q \in F]$. Note that $q \in \rho_A(q)$ and, by Lemma 3.9, for all $q' \in Q$ such that $q' \prec_A q''$ for some $q'' \in \rho_A(q)$ we have $q' \prec_A q$, and hence there exists, by induction, a series-rational expression $e_{q'} \in \mathcal{T}_\Sigma$ such that $[\![e_{q'}]\!] = L_A(q')$. So the expressions $e_{qq'}^{\rho_A(q)}$ are, indeed, defined in Definition 5.1.

▶ **Corollary 5.3.** *For every state $q \in Q$ we have $[\![e_q^+]\!] = L_A(q) - \{1\}$ and $[\![e_q]\!] = L_A(q)$.*

6 Discussion

Another automaton formalism for pomsets, *branching automata*, was proposed by Lodaya and Weil [14, 15]. Branching automata define the states where parallelism can start (*fork*) or end (*join*) in two relations; pomset automata condense this information in a single function. Lodaya and Weil also provided a translation of CKA expressions to branching automata, based on Thompson's construction [17], which relies on the fact that their automata encode transitions non-deterministically, i.e., as *relations*. Our translation, in contrast, is based on Brzozowski's construction [2], and directly constructs transition *functions* from the expressions. Lastly, their translation of branching automata to CKA expressions is only sound for a *semantically* restricted class of automata, whereas our restriction is *syntactic*.

Jipsen and Moshier [7] provided an alternative formulation of the automata proposed by Lodaya and Weil, confusingly also called *branching automata*. Their method to encode parallelism in these branching automata is conceptually dual to pomset automata: branching automata distinguish based on the target states of traces to determine the join state, whereas pomset automata distinguish based on the origin states of traces. The translations of CKA expressions to branching automata and vice versa suffer from the same shortcomings as those by Lodaya and Weil. Specifically, the translation of expressions to automata similarly relied on non-determinism, and the translation from automata to expressions only worked for definition of language that was semantically restricted along the same lines as the semantic restriction put on (the earlier definition of) branching automata by Lodaya and Weil.

7 Further work

We plan to extend our results to CKA semantics in terms of downward-closed pomset languages, i.e., sets of pomsets that are closed under Gischer's subsumption order [3]. Such an extension would correspond to adding the weak exchange law (which relates sequential and parallel compositions) to the axioms of CKA. We conjecture that our construction can be generalized to show the same correspondence between this semantics of series-rational expressions and pomset automata (i.e., without changes to the automaton model), just like Struth and Laurence suspect that the downward-closed semantics of series-parallel expressions can be captured by their non-downward closed semantics.

A classic result by Kozen [10] axiomatizes rational expressions using Kleene's theorem [9] and the uniqueness of minimal finite automata. It would be interesting to see if the same technique can be used (based on pomset automata) to axiomatize series-rational expressions. Although an axiomatization of series-rational expressions was recently published [13], this proof does not rely on an automaton model. As such, it cannot be extended to axiomatize CKA with respect to the downward-closed semantics if the above conjecture holds.

Brzozowski derivatives for classic rational expressions induce a coalgebra on rational expressions that corresponds to a finite automaton. We aim to study series-rational expressions coalgebraically. The first step would be to find the coalgebraic analogue of pomset automata such that language acceptance is characterized by the homomorphism into the final coalgebra. Ideally, such a view of pomset automata would give rise to a decision procedure for equivalence of series-rational expressions based on coalgebraic bisimulation-up-to [16].

Rational expressions can be extended with tests to reason about imperative programs equationally [12]. In the same vein, one can extend series-rational expressions with tests [6, 7] to reason about parallel imperative programs equationally. We are particularly interested in employing such an extension to extend the network specification language NetKAT [1] with primitives for concurrency so as to model and reason about concurrency within networks.

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *Proc. Principles of Programming Languages (POPL)*, pages 113–126, 2014. `doi:10.1145/2535838.2535862`.

2 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. `doi:10.1145/321239.321249`.

3 Jay L. Gischer. The equational theory of pomsets. *Theor. Comput. Sci.*, 61:199–224, 1988. `doi:10.1016/0304-3975(88)90124-7`.

4 C. A. R. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra. In *Proc. Concurrency Theory (CONCUR)*, pages 399–414, 2009. `doi:10.1007/978-3-642-04081-8_27`.

5 Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. Developments in Concurrent Kleene Algebra. *J. Log. Algebr. Meth. Program.*, 85(4):617–636, 2016. `doi:10.1016/j.jlamp.2015.09.012`.

6 Peter Jipsen. Concurrent Kleene Algebra with tests. In *Proc. Relational and Algebraic Methods in Computer Science (RAMiCS) 2014*, pages 37–48, 2014. `doi:10.1007/978-3-319-06251-8_3`.

7 Peter Jipsen and M. Andrew Moshier. Concurrent Kleene Algebra with tests and branching automata. *J. Log. Algebr. Meth. Program.*, 85(4):637–652, 2016. `doi:10.1016/j.jlamp.2015.12.005`.

8 Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. A Kleene theorem for bi-Kleene algebra. `arXiv:TODO`.

9 Stephen C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–41, 1956.

10 Dexter Kozen. A completeness theorem for Kleene Algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. `doi:10.1006/inco.1994.1037`.

11 Dexter Kozen. *Automata and computability.* Undergraduate texts in computer science. Springer, 1997.

12 Dexter Kozen. Kleene Algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, 1997. `doi:10.1145/256167.256195`.

13 Michael R. Laurence and Georg Struth. Completeness theorems for Bi-Kleene Algebras and series-parallel rational pomset languages. In *Proc. Relational and Algebraic Methods in Computer Science (RAMiCS)*, pages 65–82, 2014. `doi:10.1007/978-3-319-06251-8_5`.

14 Kamal Lodaya and Pascal Weil. Series-parallel posets: Algebra, automata and languages. In *Proc. Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 555–565, 1998. `doi:10.1007/BFb0028590`.

15 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1):347–380, 2000. `doi:10.1016/S0304-3975(00)00031-1`.

16 Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Coalgebraic bisimulation-up-to. In *Proc. Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 369–381, 2013. `doi:10.1007/978-3-642-35843-2_32`.

17 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. `doi:10.1145/363347.363387`.