On Decidability of Concurrent Kleene Algebra*†

Paul Brunet[1], Damien Pous[2], and Georg Struth[3]

1 **Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France**
2 **Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France**
3 **Department of Computer Science, The University of Sheffield, UK**

Abstract

Concurrent Kleene algebras support equational reasoning about computing systems with concurrent behaviours. Their natural semantics is given by series(-parallel) rational pomset languages, a standard true concurrency semantics, which is often associated with processes of Petri nets. We use constructions on Petri nets to provide two decision procedures for such pomset languages motivated by the equational and the refinement theory of concurrent Kleene algebra. The contribution to the first problem lies in a much simpler algorithm and an ExpSpace complexity bound. Decidability of the second, more interesting problem is new and, in fact, ExpSpace-complete.

1998 ACM Subject Classification F.3.1 Specifying, verifying, and reasoning about programs

Keywords and phrases Concurrent Kleene algebra, series-parallel pomsets, Petri nets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.28

1 Introduction

Kleene algebras axiomatise the equational theory of rational expressions. Their canonical models are rational languages and their equational theories correspond to rational expression equivalence [12, 11, 1, 2]. Deciding identities in Kleene algebras is therefore PSpace-complete [17] by standard automata constructions. Variants of Kleene algebras provide simple algebraic semantics for while-programs, and, in particular, decision procedures for these.

Pomset languages [6], on the other hand, are a widely studied model of true concurrency in which words are generalised from linear orders to partial ones. Recent applications can be found, for instance, in weak memory model verification [9]. Algebras for pomsets have been proposed first by Gischer [5] and more recently, as concurrent Kleene algebra (CKA), by Hoare et al. [8], with the aim of extending the pleasant properties of Kleene algebras into concurrency. Yet much less is known about their structure.

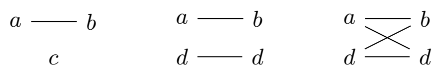
Formally, CKAs are structures $(K, +, \cdot, \|, ^\star, ^{(\star)}, 0, 1)$ that consist of a Kleene algebra $(K, +, \cdot, ^\star, 0, 1)$ and a commutative Kleene algebra $(K, +, \|, ^{(\star)}, 0, 1)$, and satisfy the weak interchange law defined below. Commutative Kleene algebras axiomatise rational commutative expression equivalence, which is decidable [4] and coNExp-complete [7]. In applications of CKA, the elements of K are typically actions of a system: The operation $+$ models nondeterministic choices, $\cdot$ and $\|$ sequential and parallel compositions, 1 the ineffective action, and 0 the abortive one. The sequential star * models the finite sequential iteration of actions

* An extended version of this abstract, including proofs, is available on HAL [3].
† This work was supported by the European Research Council (ERC) under the Horizon 2020 programme (CoVeCe, grant agreement No 678157) and the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007).

28th International Conference on Concurrency Theory (CONCUR 2017).
Editors: Roland Meyer and Uwe Nestmann; Article No. 28; pp. 28:1–28:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in terms of a least fixpoint, the parallel star $^{(\star)}$ their finite parallel iteration. It can be interpreted as the unbounded spawning of parallel processes.

Closed terms in the language of Kleene algebra correspond to rational expressions; their interpretation as word languages is standard. The extension to parallelism, hence to action-labelled partial orders and pomsets, is best explained by example. The expression $(a \cdot b) \parallel c$, for instance, is represented by the first of the following pomsets.



Execution time—the order of the poset—is indicated by lines proceeding from left to right in this implicitly directed graph. Sequential composition thus orders actions, whereas parallel composition leaves them unordered. By analogy to word languages, expressions involving $+$ or the stars require interpretations by sets of pomsets, that is, pomset languages. The expression $(a \cdot b) \parallel (c + (d \cdot d))$, for instance, denotes the language formed by the first two of the pomsets above. The third of the above pomsets is denoted by $(a \parallel d) \cdot (b \parallel d)$. It is obviously "more sequential" than the pomset to its left, which is denoted by $(a \cdot b) \parallel (d \cdot d)$. A corresponding refinement order, which compares degrees of sequentiality, has been defined on pomsets (as the smoother-than relation) by Grabowski [6]. It is isomorphic to the inclusion order on refinement-closed pomset languages and induces a (refinement) order on CKA expressions. Gischer [5] has shown that this order is characterised precisely by the inequality $(a \parallel c) \cdot (b \parallel d) \leq (a \cdot b) \parallel (c \cdot d)$ on CKA expressions (without the stars). This weak interchange law is also one of the standard CKA axioms.

Pomset languages are typically infinite when expressions contain stars. In addition, the width of individual pomsets can be unbounded when parallel stars occur; this star is therefore often omitted [15]. Furthermore, CKA expressions generate subclasses of pomset languages. Those generated by expressions over the full CKA signature are called *series-parallel-rational* (spr-languages), those generated by using a signature without the parallel star are called *series-rational* (sr-languages). Expressions are named accordingly. All pomsets occurring in spr- or sr-languages, which are built inductively from singleton pomsets by sequential and parallel compositions, are series-parallel or, equivalently, free of N-shape subpomsets [19, 6].

Equivalence of spr-expressions, as induced by spr-language identity, is decidable and can be axiomatised by any set of axioms for Kleene algebras plus those for commutative Kleene algebras [13], but a reasonable upper complexity bound has not been established. In the context of CKA with the interchange axiom, completeness or decidability of the refinement of spr-expressions, or even sr-expressions, as induced by inclusion of refinement-closed spr- or sr-languages, remains open. These questions are of obvious interest for comparing concurrent systems with respect to their degree of sequentiality or linearisability

Our first contribution consists in a simple new algorithm and a first complexity bound for sr-expression equivalence. First, using a construction similar to Thompson's [18] and Grabowski's [6], we show that every sr-language is the pomset trace language of a safe labelled Petri net. Using a result by Jategaonkar and Meyer on pomset languages of Petri nets [10], it then follows that sr-expression equivalence is in ExpSpace (Theorem 5).

Our second, more interesting contribution is a proof that sr-expression refinement is ExpSpace-complete (Theorem 26). Note that sr-expression equivalence is sr-expression refinement in both directions. This result requires comparing runs in Petri nets up-to Grabowski's refinement order, using the freedom provided by this formalism to reorder transitions, and a schedule for constructing a comparison function in a canonical way. Preservation of sequentiality or causality in this construction is somewhat intricate: it requires tracking the history and relationships between *loci* (Section 5.2 and 5.3). The Petri net approach seems natural once more due to the correspondence between nets and pomset languages, and our previous construction. Hardness of sr-expression refinement follows from a reduction from the equivalence problem for regular expressions with a shuffle operation [16], using results by Grabowski that relate pomset and shuffle languages.

2 Preliminary definitions

2.1 Pomsets

We fix a finite alphabet Σ. A *labelled poset* is a triple $\langle X, \leq, \lambda\rangle$ where X is a finite carrier set, $\leq$ is a partial order on X and the map $\lambda : X \to \Sigma$ labels every element in X with a letter in Σ. A *(labelled poset) morphism* is a function between labelled posets that preserves the order and the labels. A *pomset* is an isomorphism class of labelled posets; it is a labelled poset up-to bijective renaming of the elements in X. We represent pomsets as graphs that are implicitly directed from left to right. The vertices, which are the elements of the pomset, are labelled by λ; those edges that can be deduced by transitivity and reflexivity are omitted.

We define the following pomsets and operations on pomsets:

- The *empty pomset*, denoted by P_0, is defined as $\langle \emptyset, \emptyset, []\rangle$ ($[]$ denoting the empty function);
- for $a \in \Sigma$, the *singleton pomset* P_a is $\langle \{\bullet\}, \{\langle \bullet, \bullet\rangle\}, [\bullet \mapsto a]\rangle$;
- for pomsets $P_1 = \langle X_1, \leq_1, \lambda_1\rangle$ and $P_2 = \langle X_2, \leq_2, \lambda_2\rangle$ with $X_1 \cap X_2 = \emptyset$; the *parallel product of P_1 and P_2* is the pomset obtained by putting them side by side:

  $$P_1 \parallel P_2 \triangleq \langle X_1 \cup X_2, \leq_1 \cup \leq_2, \lambda_1 \cup \lambda_2\rangle \quad ;$$

- the *sequential product* of P_1 and P_2 is the pomset obtained by further declaring all elements of P_1 as smaller than those of P_2:

  $$P_1 ; P_2 \triangleq \langle X_1 \cup X_2, \leq_1 \cup \leq_2 \cup X_1 \times X_2, \lambda_1 \cup \lambda_2\rangle \quad .$$

Pomset P_1 *refines* P_2, written $P_1 \sqsubseteq P_2$, if there exists a bijective morphism $\varphi : X_2 \to X_1$. By definition, therefore,

- $\forall x \in X_2, \lambda_1(\varphi(x)) = \lambda_2(x)$, i.e., the bijection preserves labels; and
- $\forall x, y \in X_2, x \leq_2 y \Rightarrow \varphi(x) \leq_1 \varphi(y)$, i.e., the morphism preserves edges in P_2.

The relation $\sqsubseteq$ is a partial order on pomsets. We write ${}^{\sqsubseteq}S$ for the downward closure of a set S of pomsets with respect to it: ${}^{\sqsubseteq}S \triangleq \{P \mid \exists Q : P \sqsubseteq Q,\ Q \in S\}$. We then extend the refinement order to a preorder on sets of pomsets: $S \sqsubseteq S' \triangleq S \subseteq {}^{\sqsubseteq}S'$. (This definition is equivalent to $S \sqsubseteq S' \triangleq {}^{\sqsubseteq}S \subseteq {}^{\sqsubseteq}S'$.)

2.2 Expressions and pomset languages

A *series-rational expression*, or more briefly *expression*, is a term derived from the following syntax. The set of expressions over the alphabet Σ is written $\mathrm{Rat}^{\parallel}\langle\Sigma\rangle$.

$$e, f ::= e + f \mid e \cdot f \mid e \parallel f \mid e^\star \mid 0 \mid 1 \mid a \qquad (a \in \Sigma)$$

The *language* of an expression is the set of pomsets defined inductively as follows:

$$[\![1]\!] \triangleq \{P_0\} \quad [\![e \cdot f]\!] \triangleq \{P ; Q \mid P \in [\![e]\!],\ Q \in [\![f]\!]\} \quad [\![e \parallel f]\!] \triangleq \{P \parallel Q \mid P \in [\![e]\!],\ Q \in [\![f]\!]\} \, .$$

$$[\![0]\!] \triangleq \emptyset \quad [\![e + f]\!] \triangleq [\![e]\!] \cup [\![f]\!] \quad [\![e^\star]\!] \triangleq \bigcup_{n \in \mathbb{N}} \{P_1 ; \ldots ; P_n \mid \forall i \leq n, P_i \in [\![e]\!]\} \quad [\![a]\!] \triangleq \{P_a\} \, .$$

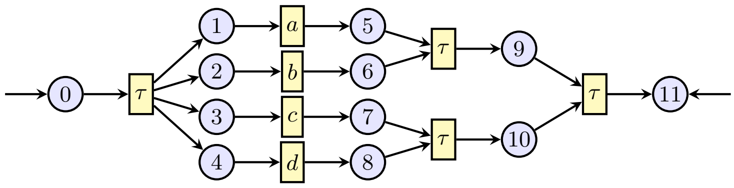
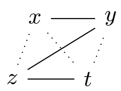


Figure 1 Example of labelled safe Petri net.

A set of pomsets is called *(series-)rational* if it is the language of some expression. It is called *downward-closed rational* if it is the downward-closure of a rational language.

Note that due to the structure of expressions, the pomsets we consider are always *series-parallel*: they are built from trivial pomsets by using sequential and parallel compositions. Valdes et al. proved that this property is equivalent to *N-freeness* [19, 6]: whenever there are four distinct elements x, y, z, t such that $x \leq y$, $z \leq y$, and $z \leq t$, then either $z \leq x$, $t \leq y$, or $x \leq t$.



In the present work we are interested in the following two decision problems.

▶ **Definition 1.** Given two expressions e, f, the problem $\mathrm{biKA}(e, f)$ asks if $[\![e]\!] \subseteq [\![f]\!]$.

▶ **Definition 2.** Given two expressions e, f, the problem $\mathrm{CKA}(e, f)$ asks if $[\![e]\!] \sqsubseteq [\![f]\!]$.

The first problem, $\mathrm{biKA}(e, f)$, asks essentially about equivalence of the sr-expressions e and f. As outlined in the introduction, axioms for Kleene algebras plus those for commutative Kleene algebras (here in fact commutative idempotent semirings without a parallel star) are complete w.r.t. this equivalence. The second one, $\mathrm{CKA}(e, f)$, asks whether e is a refinement of f, which relates to CKA with the interchange law, yet again without a parallel star. We conjecture that the aforementioned axioms together with weak interchange are complete for this semantics, but this problem remains open, to the best of our knowledge.

2.3 Labelled safe Petri nets

We now define *labelled safe Petri nets*—the machines that we use to recognise rational pomset languages. We write $\wp_+(X)$ for the set of non-empty subsets of a set X.

A *labelled Petri net* is a tuple $\mathcal{N} = \langle \mathcal{P}, T, p_{in}, p_{fin} \rangle$ where:

- $\mathcal{P}$ is a finite set of *places*;
- $T \subseteq \wp_+(\mathcal{P}) \times (\Sigma \cup \{\tau\}) \times \wp_+(\mathcal{P})$ is a set of *labelled transitions*;
- $p_{\in} \in \mathcal{P}$ is the *initial place*;
- $p_{fin} \in \mathcal{P}$ is the *final place*.

If $t = \langle P, x, P' \rangle$ is a transition, then P is its *input set*, written ${}^{\bullet}t$, x is its *label*, written $\ell(t)$, and P' is its *output set*, written $t^{\bullet}$. Transitions labelled with τ are called *silent*; the others are called *visible*. Without loss of generality, we may restrict ourselves to Petri nets where all inputs and outputs of visible transitions are singleton sets. An example of such a Petri net is displayed in Figure 1.

A *configuration* is a set of places. A transition t is *enabled* from a configuration C if ${}^{\bullet}t \subseteq C$. Whenever t is enabled in C, then firing this transition leads to the configuration

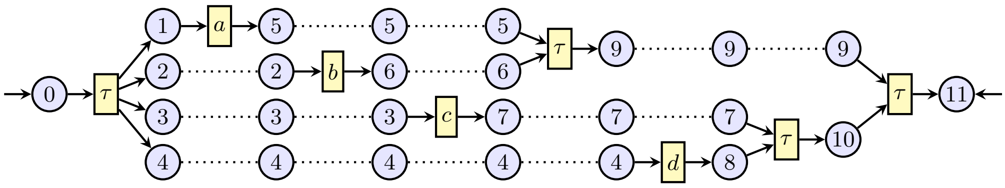


Figure 2 An accepting run of the Petri net in Figure 1.

$C' \triangleq (C \setminus {}^\bullet t) \cup t^\bullet$, and we write $C \xrightarrow{t}_{\mathcal{N}} C'$. A *run* from C_0 to C_n is a sequence $t_1; \ldots; t_n$ such that there exists configurations $C_1, \ldots, C_{n-1}$ such that

$$C_0 \xrightarrow{t_1}_{\mathcal{N}} C_1 \xrightarrow{t_2}_{\mathcal{N}} \cdots C_{n-1} \xrightarrow{t_n}_{\mathcal{N}} C_n \quad .$$

We write $C_0 \xrightarrow{t_1;\ldots;t_n}_{\mathcal{N}} C_n$ in this case. If $C_0 = \{p_{in}\}$, then the run is *initial*, if $C_n = \{p_{fin}\}$, then it is *final*, and if both conditions hold, it is *accepting*. Finally, a configuration C is *reachable* if some initial run ends in C.

Figure 2 shows an example of an accepting run. In this representation, columns of circular nodes denote the successive configurations C_i. We draw the transition t_i as a rectangular node between C_{i-1} and C_i, drawing directed edges from its inputs in C_{i-1} to its node, and to its outputs in C_i. The remaining places, those in $C_{i-1} \setminus {}^\bullet t_i$ that happen again in $C_i \setminus t_i^\bullet$, are linked with dotted lines.

A Petri net is *safe* if $(C \setminus {}^\bullet t) \cap t^\bullet = \emptyset$ holds for every reachable configuration C and every transition t enabled in C. In other words, there is always at most one token in every place of a safe Petri net. This justifies our use of sets rather than multisets for configurations a posteriori: we shall only use safe Petri nets.

The *transition automaton* $\mathcal{A}(\mathcal{N})$ of a Petri net $\mathcal{N}$ is a non-deterministic finite state automaton over the alphabet of transitions of $\mathcal{N}$; its states are configurations of $\mathcal{N}$ (i.e. subsets of $\mathcal{P}$), its initial state is $\{p_{in}\}$, its only final state is $\{p_{fin}\}$, and its transitions are the triples $\langle C, t, C' \rangle$ such that $C \xrightarrow{t}_{\mathcal{N}} C'$. Writing $\mathcal{L}(\mathcal{B})$ for the usual word language of an automaton $\mathcal{B}$, the transition automaton is defined so that we have

$$\mathcal{L}(\mathcal{A}(\mathcal{N})) \triangleq \left\{ t_1 \ldots t_n \;\middle|\; \{p_{in}\} \xrightarrow{t_1;\ldots;t_n}_{\mathcal{N}} \{p_{fin}\} \right\}.$$

2.4 Language of a Petri net

Let $R = C_0 \xrightarrow{t_1;\ldots;t_n}_{\mathcal{N}} C_n$ be a run in a Petri net $\mathcal{N}$. We define the *immediate causality relation* $\to_R \subseteq [1..n] \times [1..n]$ as

$$i \to_R j \triangleq i < j \wedge \left(\exists p \in t_i^\bullet \cap {}^\bullet t_j : \forall k,\ i < k < j \Rightarrow p \notin {}^\bullet t_k \right) \quad .$$

The *causality relation* $\leq_R$ is the reflexive transitive closure of $\to_R$. Intuitively, $i \leq_R j$ holds if t_j cannot be fired in a subrun of R without firing t_i.

For each run one can define three kinds of traces [10]. For the run fromFigure 2, these are shown in Figure 3.

- The *graph-trace* $\mathcal{G}(R)$ of R is the graph $\langle [1..n], \to_R \rangle$.
- The *transition-pomset* R is the pomset $\mathbb{T}(R) \triangleq \langle [1..n], \leq_R, \lambda_R \rangle$, where $\lambda_R(i) \triangleq \ell(t_i)$.
- The *pomset-trace* $\mathbb{P}(R)$, of R is the restriction of $\mathbb{T}(R)$ to the set $\{i \mid \ell(t_i) \in \Sigma\}$ of visible actions.

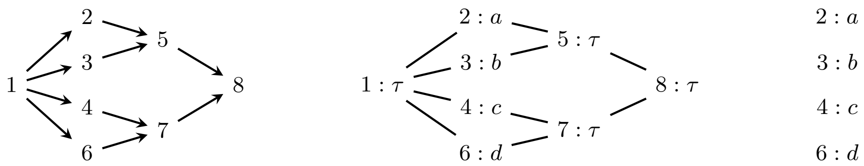


Figure 3 Graph-trace, transition-pomset, and pomset-trace of the run from Figure 2.

The *pomset language* of a Petri net $\mathcal{N}$ is the set $[\![\mathcal{N}]\!]$ of pomset-traces of accepting runs in $\mathcal{N}$. Moreover, we call a run R is *series-parallel* if its graph-trace is series-parallel. Note that this is strictly stronger than requiring that its pomset-trace be series-parallel.

The run in Figure 2 is series-parallel.

3 Reading a pomset in a Petri net

This section describes an operational way of reading and recognising pomsets with Petri nets, as one might read and recognise a word with a finite state automaton. It is independent from the rest of the paper, but might provide insight into the algorithm we develop below to compare languages of nets. Indeed, the guiding intuition behind this algorithm will be to read a net in another net.

Let $\mathcal{N} = \langle \mathcal{P}, T, p_{in}, p_{fin} \rangle$ be a safe labelled Petri net, $P = \langle X, \leq, \lambda \rangle$ a pomset, and $R = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots C_{n-1} \xrightarrow{t_n} C_n$ a run in $\mathcal{N}$. A *reading of P in $\mathcal{N}$ along R* is a sequence $\langle \rho_0, X_0 \rangle, \ldots, \langle \rho_n, X_n \rangle$ such that:

1. for every $0 \leqslant i \leqslant n$, $X_i \subseteq X$ and ρ_i is a map from C_i to $\wp(X_i)$;
2. for every $0 \leqslant i < n$,
   a. if $\ell(t_{i+1}) \in \Sigma$, and if p_0, p_1 are respectively the input and output places of t_{i+1}, there is an element $x \in \rho_i(p_0)$ such that $\lambda(x) = \ell(t_{i+1})$ and:
   $$X_{i+1} = X_i \setminus \{x\}; \qquad \rho_{i+1}(p) = \begin{cases} \{y \in X_{i+1} \mid x \leq y\} & \text{if } p = p_1 \\ \rho_{i+1}(p) = \rho_i(p) \setminus \{x\} & \text{otherwise.} \end{cases}$$
   b. if $\ell(t_{i+1}) = \tau$, then
   $$X_{i+1} = X_i; \qquad \rho_{i+1}(p) = \begin{cases} \bigcup_{q \in {}^{\bullet}t_{i+1}} \rho_i(q) & \text{if } p \in t_{i+1}^{\bullet} \\ \rho_i(p) & \text{otherwise.} \end{cases}$$

The reading is *initial* if $C_0 = \{p_{in}\}$ and $\rho_0(p_{in}) = X_0 = X$. The reading is *final* if $C_n = \{p_{fin}\}$ and $X_n = \emptyset$. The reading is *accepting* if it is both initial and final. P is *accepted* by $\mathcal{N}$ if there is an accepting reading of P in $\mathcal{N}$. The *language recognised* by $\mathcal{N}$ is the set of pomsets accepted by $\mathcal{N}$. It should not be confused with the pomset language of $\mathcal{N}$, as defined above.

▶ Remark. Notice that, if R is accepting, the existence of an accepting reading of P along R can be tested by a simple history-independent non-deterministic algorithm. We start with $X_0 = X$ and $\rho_0 = [p_{in} \mapsto X]$. At step $i+1$ we use condition 2b to compute ρ_{i+1} and X_{i+1} if t_{i+1} is silent,. If t_{i+1} is visible and there is no $x \in \rho_i({}^{\bullet}t_{i+1})$ such that $\lambda(x) = \ell(t_{i+1})$, then we conclude that there are no readings of P along R. Otherwise, we non-deterministically choose an appropriate x and use condition 2a to compute ρ_{i+1} and X_{i+1}. If this yields $X_n = \emptyset$ we have obtained an accepting reading, otherwise we can conclude that there are no such readings.

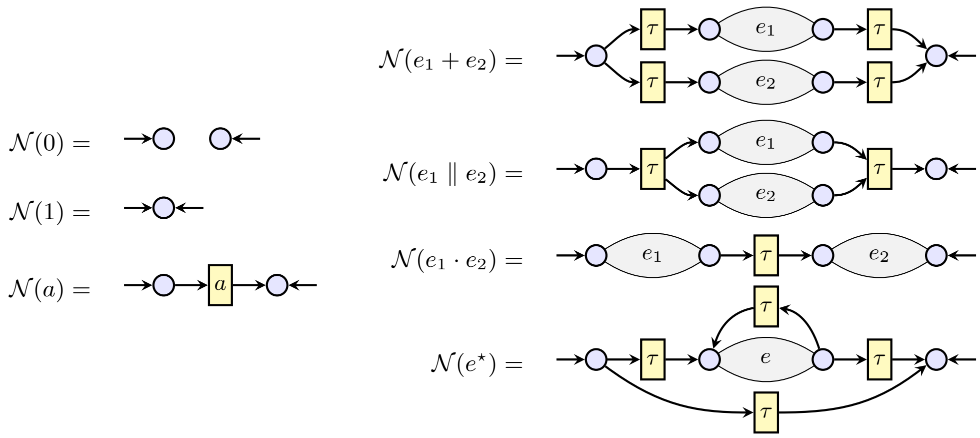
▶ **Lemma 3.** *If R is accepting, there is an accepting reading of P along R if and only if $P \sqsubseteq \mathbb{P}(R)$.*

Proof. See [3]. ◀

▶ **Corollary 4.** *The language recognised by $\mathcal{N}$ is ${}^{\sqsubseteq}[\![\mathcal{N}]\!]$.*

4 Rational Petri nets

This section shows that every rational pomset language is the pomset language of a (safe labelled) Petri net. To this end, we recursively associate with every expression e a Petri net $\mathcal{N}(e)$ such that $[\![\mathcal{N}(e)]\!] = [\![e]\!]$. Moreover, all accepting runs of this Petri net turn out to be series-parallel. The construction poses no difficulty; it is a simple adaptation of Thompson's construction for rational word languages [18], and an extension of a previous construction by Grabowski [6] for safe Petri nets and pomset languages. We only present this construction graphically here.



This construction yields decidability of biKA in exponential space. Indeed we may build the Petri nets $\mathcal{N}(e)$ and $\mathcal{N}(f)$ from the expressions e and f (these are linear in the size of e and f) and use Jategaonkar and Meyer's result [10] that testing containment of pomset-trace languages of two Petri nets is an ExpSpace-complete problem.

▶ **Theorem 5.** *The problem* biKA *lies in the class* ExpSpace.

▶ **Proposition 6.** *The language recognised by $\mathcal{N}(e)$ is ${}^{\sqsubseteq}[\![e]\!]$.*

Proof. By construction we have $[\![\mathcal{N}(e)]\!] = [\![e]\!]$. We conclude using Corollary 4. ◀

5 Comparing Petri nets modulo refinement

Next we show how to compare Petri nets modulo refinement. Thanks to the previous construction, this leads to decidability of the problem CKA. We fix two Petri nets $\mathcal{N}_1$ and $\mathcal{N}_2$ for this section and the following one. Our goal is to check whether $[\![\mathcal{N}_1]\!] \sqsubseteq [\![\mathcal{N}_2]\!]$, i.e., whether for each run $R_1 \in \mathcal{L}(\mathcal{N}_1)$, there exists a corresponding run $R_2 \in \mathcal{L}(\mathcal{N}_2)$ such that $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$.

The first difficulty is that we may have to reorder runs in $\mathcal{N}_2$: due to concurrency, transitions might be triggered in different orders and still yield the same pomset.

5.1 Reordering runs

Let $R = t_1; \ldots; t_n$ be a run from C_0 to C_n. Let π be a permutation of $[1..n]$. The action of π on R is defined as $\pi R \triangleq t_{\pi(1)}; \ldots; t_{\pi(n)}$. The permutation π is *compatible* with R if it is order-preserving:

$$\forall i, j,\ i \leq_R j \Rightarrow \pi(i) \leqslant \pi(j).$$

▶ **Lemma 7.** *If π is compatible with R, then $C_0 \xrightarrow{\pi R}_{\mathcal{N}} C_n$, $\mathcal{G}(R) = \mathcal{G}(\pi R)$, and*

$$i \leq_R j \Leftrightarrow \pi(i) \leq_{\pi R} \pi(j).$$

Proof. We can exchange two successive transitions that are not causally linked without changing the graph (up-to isomorphism). We repeat this process until we obtain πR. ◀

Accordingly, we say that a run R' is *equivalent to a run* R if $R' = \pi R$ for some compatible permutation π.

Another important notion for the completeness of the method we propose is that of an *economical* run: a run that fires its silent transitions as late as possible.

▶ **Definition 8.** A run $t_1; \ldots; t_n$ is *economical* if for all $i < j$, if $t_i, \ldots, t_{j-1}$ are silent transitions and t_j is a visible transition, then $i \leq_R j$.

The run in Figure 2 is not economical: the fifth transition is a silent one, but it is not causally related to the next transition, which is visible. We will see that it can be reordered into an equivalent economical run (Proposition 10 and Example 11 below.)

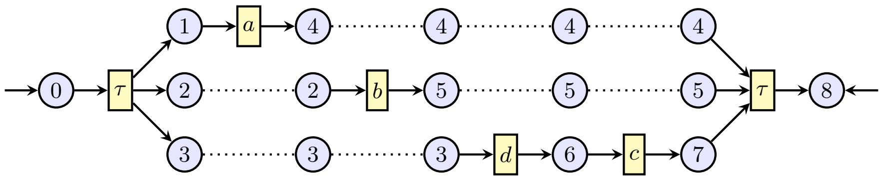
Even more importantly when comparing two runs, we need to ensure that the visible transitions are fired in the same order.

▶ **Definition 9.** Given a run R_1 in $\mathcal{N}_1$ and a run R_2 in $\mathcal{N}_2$ such that $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$, we say that R_2 *follows* R_1 if the subsumption is witnessed by a bijection φ such that for every two visible indices i, j in R_2 we have $i < j \Leftrightarrow \varphi(i) < \varphi(j)$.

▶ **Proposition 10.** *Let R_1 and R_2 be series-parallel runs in $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. If $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$ then there exists an economical and series-parallel run R_2' in $\mathcal{N}_2$ that follows R_1 and is equivalent to R_2.*

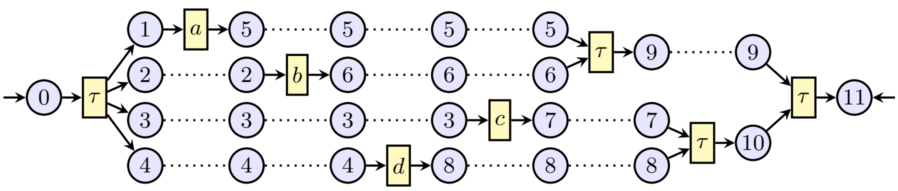
▶ **Example 11.** Consider the run R_2 from Figure 2 and the following run R_1:

R_1 :



The pomset of R_1 is $\mathbb{P}(R_1) = P_a \parallel P_b \parallel (P_d; P_c)$, and we may check that $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$. To transform R_2 into a run that is economical and follows R_1, we must (1) exchange the transitions labelled with c and d; and (2) delay the silent transition in the middle of R_2 until all visible transitions have been fired. Doing so, we get the following run R_2', which follows R_1 and is equivalent to R_2.

R_2' :



5.2 Loci

In this more technical section, we define *loci*, as a way to lift the causality relation between transitions to places in the successive configurations of the runs. Let R be a run, with $R = C_0 \xrightarrow{t_1} C_1 \cdots C_{n-1} \xrightarrow{t_n} C_n$, $\mathbb{T}(R) = \langle [1..n], \leq_R, \lambda \rangle$. A *locus* denotes a pair $\langle p, i \rangle$ where $0 \leqslant i \leqslant n$ and $p \in C_i$. In some sense, loci are places with a time index. In the previous pictures those are the numbered circles: p is the number in the circle (the name of the place), and i is the index of its column. Formally, the set of loci of the run R is $\bigcup_{0 \leqslant i \leqslant n} C_i \times \{i\}$. We generate an equivalence relation $\approx_R$ on loci using the following rule:

$$p \notin {}^\bullet t_i \Rightarrow \langle p, i \rangle \approx_R \langle p, i-1 \rangle .$$

Graphically, equivalent loci are linked with dotted lines. Equivalences classes with respect to $\approx_R$ are thus places with a time interval, rather than a single index. The *source* of a locus $\langle p, i \rangle$ is the smallest index of its equivalence class:

$$src \langle p, i \rangle \triangleq \min \{ j \leqslant i \mid \langle p, i \rangle \approx_R \langle p, j \rangle \} \quad .$$

Now we define a preorder $\precsim_R$, generated by the following rules:

$$p, q \in {}^\bullet t_i \times t_i^\bullet \Rightarrow \langle p, i-1 \rangle \precsim_R \langle q, i \rangle \quad , \qquad \langle p, i \rangle \approx_R \langle q, j \rangle \Rightarrow \langle p, i \rangle \precsim_R \langle q, j \rangle \quad .$$

Note that two loci in the same configuration are always incomparable. Finally, we inductively define the set of indices of *predecessors* of a locus:

- $pred \langle p, 0 \rangle \triangleq \emptyset$.
- if $p \in t_i^\bullet$, then $pred \langle p, i \rangle \triangleq \{i\} \cup \bigcup_{q \in {}^\bullet t_i} pred \langle q, i-1 \rangle$.
- if $p \notin t_i^\bullet$, then $pred \langle p, i \rangle \triangleq pred \langle p, i-1 \rangle$.

The *set of visible predecessors of a locus*, written $vpred \langle p, i \rangle$, is the subset of indices of visible transitions in $pred \langle p, i \rangle$.

▶ **Lemma 12.** *The following properties hold:*

$$\forall i, \forall p \in t_i^\bullet,\ pred \langle p, i \rangle = \{ j \mid j \leq_R i \}, \tag{1}$$
$$\forall i, \forall p \in C_i,\ pred \langle p, i \rangle = \{ j \mid j \leq_R src \langle p, i \rangle \}, \tag{2}$$
$$\forall i, j, p, q,\ \langle p, i \rangle \approx_R \langle q, j \rangle \Rightarrow pred \langle p, i \rangle = pred \langle q, j \rangle, \tag{3}$$
$$\forall i, j, p, q,\ \langle p, i \rangle \precsim_R \langle q, j \rangle \Rightarrow pred \langle p, i \rangle \subseteq pred \langle q, j \rangle, \tag{4}$$
$$\forall i, j, p, \forall q \in t_j^\bullet,\ j \in vpred \langle p, i \rangle \Rightarrow \langle q, j \rangle \precsim_R \langle p, i \rangle, \tag{5}$$
$$\forall i \leqslant j,\ \exists p \in C_j : i \in pred \langle p, j \rangle . \tag{6}$$

Notice that we managed to lift the causality relation $\leq_R$ to the level of loci: this lemma implies that if $i \neq j$, $p \in t_i^\bullet$ and $q \in t_j^\bullet$, then $i \leq_R j$ if and only if $\langle p, i \rangle \precsim_R \langle q, j \rangle$.

5.3 Schedules

We compare runs using the following notion of *schedule*, where we interleave two runs in such a way that they synchronise on visible transitions.

▶ **Definition 13.** Let R_1 and R_2 be two runs with $R_i = C_0^i \xrightarrow{t_1^i} C_1^i \cdots C_{n_i-1}^i \xrightarrow{t_{n_i}^i} C_{n_i}^i$ for $i \in \{1, 2\}$. An N-*schedule* from R_1 to R_2 is a function $\eta : [0..N] \to [0..n_1] \times [0..n_2]$ such that

- $\eta(0) = \langle 0, 0 \rangle$ and $\eta(N) = \langle n_1, n_2 \rangle$;
- if $\eta(k) = \langle i, j \rangle$, then either

1. t^1_{i+1} is a silent transition and $\eta(k+1) = \langle i+1, j\rangle$, or
2. t^2_{j+1} is a silent transition and $\eta(k+1) = \langle i, j+1\rangle$, or
3. t^1_{i+1} and t^2_{j+1} are visible, $\ell\left(t^1_{i+1}\right) = \ell\left(t^2_{j+1}\right)$ and $\eta(k+1) = \langle i+1, j+1\rangle$.

Note that a schedule constructs a bijection between the visible transitions of R_1 and those of R_2. Indeed, each of these transitions must be fired synchronously and agree on labels (case 3). Furthermore, since a schedule starts with $\eta(0) = \langle 0, 0\rangle$ and ends with $\eta(N) = \langle n_1, n_2\rangle$, every transition in both runs must be fired. This means there is a label-preserving bijection φ, called *the bijection induced by* η, from the visible transitions of R_2 to those of R_1 that satisfies $i < j$ if and only if $\varphi(i) < \varphi(j)$.

The notion of schedule is still very weak: there are schedules from one run to another whenever they have the same visible transitions, in the same order. Causality between those transitions is not taken into account. We fix this with the following technical definition. Intuitively, we keep track of the history and relationships between the loci of the configurations, in order to ensure that the causality relation in the presumably smaller run R_1 refines that of R_2.

▶ **Definition 14.** For each N-schedule we define the following sequence of binary relations $(\prec_k)_{k\in[0..N]}$, where $\prec_k \subseteq C^1_i \times C^2_j$ when $\eta(k) = \langle i, j\rangle$, by induction:

- $\prec_0 = C^1_0 \times C^2_0$;
- if $\eta(k) = \langle i, j\rangle$, then
  1. if $\eta(k+1) = \langle i+1, j\rangle$, we set $p \prec_{k+1} q \triangleq \begin{cases} p \prec_k q & \text{if } p \notin (t^1_{i+1})^\bullet, \\ \exists p' \in {}^\bullet t^1_{i+1}, p' \prec_k q & \text{otherwise.} \end{cases}$
  2. if $\eta(k+1) = \langle i, j+1\rangle$, we set $p \prec_{k+1} q \triangleq \begin{cases} p \prec_k q & \text{if } q \notin (t^2_{j+1})^\bullet, \\ \forall q' \in {}^\bullet t^2_{j+1} : p \prec_k q' & \text{otherwise.} \end{cases}$
  3. otherwise, let $t^1_{i+1} = \langle \{p_0\}, a, \{p_1\}\rangle$ and $t^2_{j+1} = \langle \{q_0\}, a, \{q_1\}\rangle$; we set

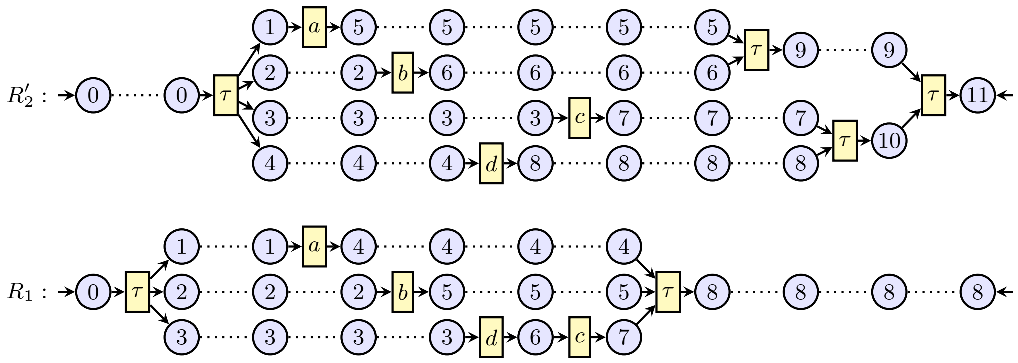
$$p \prec_{k+1} q \triangleq \begin{cases} p_0 \prec_k q \text{ or } (q = q_1 \text{ and } p_0 \prec_k q_0) & \text{if } p = p_1 \\ p \prec_k q \text{ and } q \neq q_1 & \text{otherwise.} \end{cases}$$

The schedule η is *valid* if for every visible index i in R_2 we have $p \prec_k q$ for the unique k, p, q such that $\eta(k) = \langle \varphi(i), i\rangle$, $(t^1_{\varphi(i)})^\bullet = \{p\}$ and $(t^2_i)^\bullet = \{q\}$.

▶ **Example 15.** Recall the runs R'_2 and R_1 from Example 11. The following sequence is a schedule from R_1 to R'_2.

$$\eta = (0,0); (1,0); (1,1); (2,2); (3,3); (4,4); (5;5); (6,5); (6,6); (6,7); (6,8)$$

We may then draw the two runs side by side according to this schedule:



From this schedule, we can compute the successive $\prec_k$ relations, and check that $4 \prec_3 5$, $5 \prec_4 6$, $6 \prec_5 8$, and $7 \prec_6 7$. Hence η is valid.

▶ Remark. If $\eta(k) = \langle i, j\rangle$, $\eta(k') = \langle i', j'\rangle$, $\langle p, i\rangle \approx_{R_1} \langle p', i'\rangle$ and $\langle q, j\rangle \approx_{R_2} \langle q', j'\rangle$, then $p \prec_k q$ if and only if $p' \prec_{k'} q'$.

▶ **Lemma 16.** *If $\eta(k) = \langle i, j\rangle$, then $p \prec_k q$ entails $\varphi(vpred\,\langle q, j\rangle) \subseteq vpred\,\langle p, i\rangle$.*

The algorithm we define in the next section looks for valid schedules. The following proposition establishes soundness of this strategy.

▶ **Proposition 17.** *If there exists a valid schedule from R_1 to R_2, then $\mathbb{P}\,(R_1) \sqsubseteq \mathbb{P}\,(R_2)$.*

Proof. The bijection φ induced by η works. Let i, j be visible indices in R_2 such that $i \leq_{R_2} j$. We need to show that $\varphi(i) \leq_{R_1} \varphi(j)$. Take the unique k, p, q such that $\eta(k) = \langle\varphi(j), j\rangle$, $(t^1_{\varphi(j)})^\bullet = \{p\}$ and $(t^2_j)^\bullet = \{q\}$. By Lemma 12.(1) we have $i \in vpred\,\langle q, j\rangle$. Since η is valid, we have $p \prec_k q$, and thus $\varphi(vpred\,\langle q, j\rangle) \subseteq vpred\,\langle p, \varphi(j)\rangle$ by Lemma 16. This means that $\varphi(i) \in vpred\,\langle p, \varphi(j)\rangle$, and using Lemma 12.(1) again we obtain $\varphi(i) \leq_{R_1} \varphi(j)$. ◀

For completeness of the algorithm we need to exhibit valid schedules. Under appropriate assumptions—see Proposition 19 below—the following *canonical schedule* $\boldsymbol{\eta}$ from R_1 to R_2 will work. We define it recursively. Intuitively, we schedule the silent transitions of R_1 as early as possible and those of R_2 as late as possible:

- $\boldsymbol{\eta}(0) = \langle 0, 0\rangle$;
- if $\boldsymbol{\eta}(k) = \langle i, j\rangle$ then
  1. if t^1_{i+1} is silent, then $\boldsymbol{\eta}(k + 1) = \langle i + 1, j\rangle$;
  2. if t^1_{i+1} is visible and t^2_{j+1} is silent then $\boldsymbol{\eta}(k + 1) = \langle i, j + 1\rangle$;
  3. if t^1_{i+1} and t^2_{j+1} are visible, then $\boldsymbol{\eta}(k + 1) = \langle i + 1, j + 1\rangle$.

We write $\boldsymbol{\varphi}$ for the bijection induced by $\boldsymbol{\eta}$. The schedule from Example 15 is actually the canonical schedule. The converse of Lemma 16 holds for the canonical schedule:

▶ **Lemma 18.** *If R_2 is series-parallel and economical, then, for every k with $\boldsymbol{\eta}(k) = \langle i, j\rangle$, if $\boldsymbol{\varphi}(vpred\,\langle q, j\rangle) \subseteq vpred\,\langle p, i\rangle$, then $p \prec_k q$.*

▶ **Proposition 19.** *If R_1 and R_2 are series-parallel, if $\mathbb{P}\,(R_1) \sqsubseteq \mathbb{P}\,(R_2)$, and if R_2 is economical and follows R_1 then the canonical schedule $\boldsymbol{\eta}$ from R_1 to R_2 is valid.*

Proof. First note that since R_2 follows R_1, $\boldsymbol{\varphi}$ and the bijection witnessing $\mathbb{P}\,(R_1) \sqsubseteq \mathbb{P}\,(R_2)$ must coincide. Let i be a visible index in R_2, and let k, p, q such that $\boldsymbol{\eta}(k) = \langle\boldsymbol{\varphi}(i), i\rangle$, $(t^1_{\boldsymbol{\varphi}(i)})^\bullet = \{p\}$ and $(t^2_i)^\bullet = \{q\}$. We have to prove $p \prec_k q$. By Lemma 18, it suffices to prove the inclusion $\boldsymbol{\varphi}(vpred\,\langle q, i\rangle) \subseteq vpred\,\langle p, \boldsymbol{\varphi}(i)\rangle$, i.e., that for every $j \in vpred\,\langle q, i\rangle$, we have $\boldsymbol{\varphi}(j) \in vpred\,\langle p, \boldsymbol{\varphi}(i)\rangle$. This is equivalent to checking that for every visible j, $j \leq_{R_2} i$ implies $\boldsymbol{\varphi}(j) \leq_{R_1} \boldsymbol{\varphi}(i)$, which is true because $\boldsymbol{\varphi}$ is an order preserving bijection from the pomsets of R_2 to that of R_1. ◀

Note that this lemma relies on the fact that R_2 is series-parallel. Indeed, there can be pairs of runs R_1 and R_2 satisfying $\mathbb{P}\,(R_1) \sqsubseteq \mathbb{P}\,(R_2)$, and R_2 being economical and following R_1, but such that the canonical schedule is not valid.

5.4 Reduction to finite automata

Now that we have the notion of valid schedule, we use a technique similar to [10] to reduce the problem of comparing Petri nets modulo subsumption to the comparison of plain automata. For this end, we define the following automaton that aims at recognising those runs of $\mathcal{N}_1$ for which there exists a valid schedule to some run in $\mathcal{N}_2$.

▶ **Definition 20.** The *composite automaton* $\mathcal{N}_1 \prec \mathcal{N}_2$ is the nondeterministic finite state automaton with epsilon-transitions $\langle Q, T_1, q_0, F\rangle$ where:

- the alphabet is the set of transitions of $\mathcal{N}_1$;
- the set of states Q consists of triples $\langle C_1, C_2, \prec\rangle$ with C_1 and C_2 respectively configurations of $\mathcal{N}_1$ and $\mathcal{N}_2$ and $\prec \subseteq C_1 \times C_2$;
- the initial state q_0 is the triple $\left\langle \left\{p^1_{in}\right\}, \left\{p^2_{in}\right\}, \left\{\left\langle p^1_{in}, p^2_{in}\right\rangle\right\}\right\rangle$;
- final states are those triples of the shape $\left\langle \{p^1_{fin}\}, \{p^2_{fin}\}, \_\right\rangle$;
- transitions are split into three kinds:
  1. if t is a silent transition of $\mathcal{N}_1$, $C_1 \xrightarrow{t}_{\mathcal{N}_1} C_1'$, then from every state $\langle C_1, C_2, \prec\rangle$ there is a transition labelled with t going to the state $\langle C_1', C_2, \prec'\rangle$ with
$$p \prec' q \Leftrightarrow \begin{cases} p \prec q & \text{if } p \notin t^\bullet, \\ \exists p' \in {}^\bullet t, p' \prec q & \text{otherwise.} \end{cases}$$
  2. if t is a silent transition of $\mathcal{N}_2$, $C_2 \xrightarrow{t}_{\mathcal{N}_2} C_2'$, then from every state $\langle C_1, C_2, \prec\rangle$ there is an epsilon-transition going to the state $\langle C_1, C_2', \prec'\rangle$ with
$$p \prec' q \Leftrightarrow \begin{cases} p \prec q & \text{if } q \notin t^\bullet, \\ \forall q' \in {}^\bullet t, p \prec q' & \text{otherwise.} \end{cases}$$
  3. if t_1 and t_2 are visible transitions of $\mathcal{N}_1$ and $\mathcal{N}_2$ with the same label, inputs p_0 and q_0 and outputs p_1 and q_1, if $C_1 \xrightarrow{t_1}_{\mathcal{N}_1} C_1'$ and $C_2 \xrightarrow{t_2}_{\mathcal{N}_2} C_2'$, then from every state $\langle C_1, C_2, \prec\rangle$ such that $p_0 \prec q_0$, there is a transition labelled with t_1 going to the state $\langle C_1', C_2', \prec'\rangle$ with
$$p \prec' q \Leftrightarrow \begin{cases} p_0 \prec q \text{ or } q = q_1 & \text{if } p = p_1, \\ p \prec q \text{ and } q \neq q_1 & \text{otherwise.} \end{cases}$$

By definition of this composite automaton, we have

▶ **Lemma 21.** *The language of the automaton $\mathcal{N}_1 \prec \mathcal{N}_2$ is the set of accepting runs R_1 in $\mathcal{N}_1$ such that there is an accepting run R_2 in $\mathcal{N}_2$ and a valid schedule from R_1 to R_2.*

Finally, we can reduce the comparison of Petri nets modulo subsumption to that of (word) automata.

▶ **Proposition 22.** *If the runs in $\mathcal{N}_1$ and those in $\mathcal{N}_2$ are all series-parallel, then $\mathcal{L}(\mathcal{A}(\mathcal{N}_1)) \subseteq \mathcal{L}(\mathcal{N}_1 \prec \mathcal{N}_2)$ if and only if $[\![\mathcal{N}_1]\!] \sqsubseteq [\![\mathcal{N}_2]\!]$.*

Proof. Suppose $\mathcal{L}(\mathcal{A}(\mathcal{N}_1)) \subseteq \mathcal{L}(\mathcal{N}_1 \prec \mathcal{N}_2)$ and let $P \in [\![\mathcal{N}_1]\!]$. There exists $R_1 \in \mathcal{L}(\mathcal{A}(\mathcal{N}_1))$ such that $P = \mathbb{P}(R_1)$. By assumption we also have $R_1 \in \mathcal{L}(\mathcal{N}_1 \prec \mathcal{N}_2)$, which means, by Lemma 21, that there is an accepting run R_2 in $\mathcal{N}_2$ and a valid schedule η from R_1 to R_2. Proposition 17 then tells us that $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$, thus proving $P \in {}^{\sqsubseteq}[\![\mathcal{N}_2]\!]$.

Conversely, assume that $[\![\mathcal{N}_1]\!] \sqsubseteq [\![\mathcal{N}_2]\!]$. Let $R_1 \in \mathcal{L}(\mathcal{A}(\mathcal{N}_1))$ be an accepting run in $\mathcal{N}_1$. By assumption, there is an accepting run R_2 in $\mathcal{N}_2$ such that $\mathbb{P}(R_1) \sqsubseteq \mathbb{P}(R_2)$. By hypothesis, both R_1 and R_2 are series-parallel. By Proposition 10, there exists an economical series-parallel run R_2' that follows R_1 and is equivalent to R_2. Hence using Proposition 19, there is a valid schedule from R_1 to R_2'. With Lemma 21 we conclude that $R_1 \in \mathcal{L}(\mathcal{N}_1 \prec \mathcal{N}_2)$. ◀

6 Decidability & Complexity of CKA

Putting together the results from Sections 4 and 5 yields the announced algorithm.

▶ **Proposition 23.** CKA *lies in the class* ExpSpace.

Proof. We build the Petri nets $\mathcal{N}(e)$ and $\mathcal{N}(f)$, and then the finite automata $\mathcal{A}(\mathcal{N}(e))$ and $\mathcal{N}(e) \prec \mathcal{N}(f)$. By Proposition 22, to answer the original question, we simply need to test these automata for language inclusion. It is well known that this requires polynomial space with respect to the size of the automata.

Let n be the size of e and m the size of f (their number of symbols). The Petri nets $\mathcal{N}(e)$ and $\mathcal{N}(f)$ are linear in the size of e and f, with at most $2n$ and $2m$ places. The automaton $\mathcal{A}(\mathcal{N}(e))$ uses at most 2^{2n} states (recall these are sets of places). The automaton $\mathcal{N}(e) \prec \mathcal{N}(f)$ uses at most $2^{2n} \times 2^{2m} \times 2^{2n\times 2m}$ states. Hence testing language equivalence of these two automata will use an amount of space polynomial in 2^{2n} and $2^{2n+2m+4nm}$, whence the announced result. ◀

For the lower bound, we reduce the problem of universality of regular expressions with shuffle [16] to the containment of downward-closed rational languages. We briefly recall the former. *Regular expressions with shuffle* over the alphabet Σ are terms over the syntax

$$e, f \in \mathrm{Rat}^{\bowtie}\langle\Sigma\rangle ::= e + f \mid e \cdot f \mid e \bowtie f \mid e^\star \mid 0 \mid 1 \mid a \qquad (a \in \Sigma).$$

Given two words u and v over Σ, the *shuffle product* of u and v, written $u \bowtie v$, is the language of all words of the form $u_1v_1u_2v_2\cdots u_kv_k$; where $u = u_1\cdots u_k$, $v = v_1\cdots v_k$, and the words u_i, v_i can be of arbitrary length (including the empty word).

The *language of a regular expression with shuffle* is defined recursively as follows.

$$[\![0]\!] := \emptyset \qquad [\![1]\!] := \{\varepsilon\} \qquad [\![a]\!] := \{a\} \qquad [\![e+f]\!] := [\![e]\!] \cup [\![f]\!] \qquad [\![e\cdot f]\!] := [\![e]\!] \cdot [\![f]\!]$$

$$[\![e \bowtie f]\!] := \bigcup_{u\in[\![e]\!],\ v\in[\![f]\!]} u \bowtie v \qquad [\![e^\star]\!] := \bigcup_{n\in\mathbb{N}} \{u_1 \dots u_n \mid \forall i \leqslant n, u_i \in [\![e]\!]\}.$$

▶ **Theorem 24** (Mayer and Stockmeyer [16]). *The problem of testing whether the language of a regular expression with shuffle is equal to $\Sigma^\star$ is* ExpSpace*-complete.*

The key observations for our reduction are due to Grabowski [6]: words are isomorphic to totally ordered pomsets, and given two words u and v, the set of totally ordered pomsets in ${}^{\sqsubseteq}[\![u \parallel v]\!]$ is isomorphic to the shuffle product of u and v.

Concretely, we associate a series-parallel expression $\lfloor e \rfloor$ to any regular expression with shuffle e by replacing every occurrence of $\bowtie$ with $\parallel$. This encoding has the following property.

▶ **Lemma 25.** *For every word $w \in \Sigma^\star$, we have $w \in [\![e]\!]$ if and only if w seen as a totally ordered pomset is in ${}^{\sqsubseteq}[\![\lfloor e \rfloor]\!]$.*

Proof. By a simple induction on e, using the above observation for the shuffle case. (Each subcase can be found in [6].) ◀

As a consequence, the language of e is $\Sigma^\star$ if and only if $[\![\Sigma^\star]\!] \sqsubseteq [\![\lfloor e \rfloor]\!]$. We thus have a linear encoding of the universality of regular expressions with shuffle into containment of downward-closed rational languages, hence our final theorem.

▶ **Theorem 26.** *The problem* CKA *is* ExpSpace*-complete.*

An implementation of the algorithm is available at `http://paul.brunet-zamansky.fr/cka.html`.

7 Related work

Several constructions in the literature are similar to those presented in Section 4. Here we list some of them, highlighting the differences between these developments and our own.

Lodaya and Weil introduced *branching automata* that recognise series-parallel rational pomset languages [15], which include the series-rational languages we use here. These automata impose a strong notion of bracketing (opening and closing τ-transitions must match exactly), which we do not know how to handle when it comes to comparing automata. This is why we used plain Petri nets instead.

Jategaonkar and Meyer presented a construction almost equivalent to ours [10], albeit for different purposes: their goal was to obtain a lower complexity bound by a reduction from the universality problem of regular languages with shuffle. The main differences in the constructions are that we use an initial *place* instead of an initial *marking*, and that we consider a unique final *place* while they have a distinguished final *transition*. These differences mainly impact the star and parallel product constructs. Jategaonkar and Meyer's construction could in fact be adapted to obtain an alternative proof of Theorem 5. However, their construction does not satisfy the structural constraints needed for the completeness of the algorithm we develop in Section 5: the runs of the automata produced by their construction are not always series-parallel.

Finally, a third construction that produces safe Petri nets from expressions was developed by Lodaya [14]. It is, however, quite different from the present approach. In particular, it requires initial and final markings, and it is not appropriate for a precise complexity analysis, as it produces nets that are exponentially large with respect to input expressions.

References

1. Maurice Boffa. Une remarque sur les systèmes complets d'identités rationnelles. *Informatique Théorique et Applications*, 24:419–428, 1990. URL: `http://archive.numdam.org/article/ITA_1990__24_4_419_0.pdf`.
2. Maurice Boffa. Une condition impliquant toutes les identités rationnelles. *Informatique Théorique et Applications*, 29(6):515–518, 1995.
3. Paul Brunet, Damien Pous, and Georg Struth. On decidability of concurrent Kleene algebra, 2017. Full version of this extended abstract, with proofs. URL: `https://hal.archives-ouvertes.fr/hal-01558108/`.
4. J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.
5. Jay L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2):199–224, 1988. `doi:10.1016/0304-3975(88)90124-7`.
6. J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4:427–498, 1981.
7. Christoph Haase and Piotr Hofman. Tightening the complexity of equivalence problems for commutative grammars. In *STACS*, volume 47 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
8. T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.
9. A. Horn and D. Kroening. On partial order semantics for SAT/SMT-based symbolic encodings of weak memory concurrency. In *FORTE*, volume 9039 of *Lecture Notes in Computer Science*, pages 19–34. Springer Verlag, 2015.
10. Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996. `doi:10.1016/0304-3975(95)00132-8`.

11 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *LICS*, pages 214–225. IEEE, 1991. `doi:10.1109/LICS.1991.151646`.

12 Daniel Krob. A Complete System of B-Rational Identities. In *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 60–73. Springer Verlag, 1990. `doi:10.1007/BFb0032022`.

13 Michael R. Laurence and Georg Struth. Completeness theorems for bi-Kleene algebras and series-parallel rational pomset languages. In *RAMiCS*, volume 8428 of *Lecture Notes in Computer Science*, pages 65–82. Springer Verlag, 2014. `doi:10.1007/978-3-319-06251-8_5`.

14 K. Lodaya, D. Ranganayakulu, and K. Rangarajan. Hierarchical structure of 1-safe petri nets. In *ASIAN*, Progamming Languages and Distributed Computation, pages 173–187. Springer Verlag, 2003. `doi:10.1007/978-3-540-40965-6_12`.

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 Alain J. Mayer and Larry J. Stockmeyer. The complexity of word problems – this time with interleaving. *Information and Computation*, 115(2):293–311, 1994. `doi:10.1006/inco.1994.1098`.

17 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT*, pages 125–129. IEEE, 1972. `doi:10.1109/SWAT.1972.29`.

18 K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11:419–422, 1968. URL: `http://www.fing.edu.uy/inco/cursos/intropln/material/p419-thompson.pdf`.

19 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *STOC*, pages 1–12. ACM, 1979. `doi:10.1145/800135.804393`.