On Decidability of Concurrent Kleene Algebra
Paul Brunet, Damien Pous, Georg Struth

To cite this version:
Paul Brunet, Damien Pous, Georg Struth. On Decidability of Concurrent Kleene Algebra. CONCUR, Sep 2017, Berlin, Germany. 2017. <hal-01558108>

HAL Id: hal-01558108
https://hal.archives-ouvertes.fr/hal-01558108
Submitted on 7 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On Decidability of Concurrent Kleene Algebra*

Paul Brunet¹, Damien Pous¹, and Georg Struth²

¹ Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France
² 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.

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

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 \((K, +, \cdot, \parallel, ^*, (\star), 0, 1)\) that consist of a Kleene algebra \((K, +, \cdot, ^*, 0, 1)\) and a commutative Kleene algebra \((K, +, \parallel, (\star), 0, 1)\), 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 \(\parallel\) sequential and parallel compositions, \(1\) the ineffective action, and \(0\) the abortive one. The sequential star \(^*\) models the finite sequential iteration of actions 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 \((a \cdot b) \parallel c, \)

---

* Extended version of the abstract in Proc. CONCUR 2017 [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)
for instance, is represented by the first of the following pomsets.

\[
\begin{array}{ccc}
a & \longrightarrow & b \\
c \\
\end{array} 
\quad 
\begin{array}{ccc}
a & \longrightarrow & b \\
d & \longrightarrow & d \\
\end{array} 
\quad 
\begin{array}{ccc}
a & \rightarrownw & b \\
d & \rightarrownw & d \\
\end{array} 
\]

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 poset 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 \(\text{ExpSpace}\) (Theorem 5).

Our second, more interesting contribution is a proof that sr-expression refinement is \(\text{ExpSpace}\)-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 \(\text{loci}\) (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 $\Sigma$. A labeled 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 $\Sigma$. A (labeled poset) morphism is a function between labeled posets that preserves the order and the labels. A pomset is an isomorphism class of labelled posets; it is a labeled 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 $\lambda$; 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 $\langle \{\bullet\}, \{\bullet, \bullet\}, \bullet \mapsto a\rangle$;
- 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:

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

- 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 $P_2$:

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

Pomset $P_1$ refines $P_2$, written $P_1 \subseteq P_2$, 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 $\subseteq$ is a partial order on pomsets. We write $\sqsubseteq S$ for the downward closure of a set $S$ of pomsets with respect to it: $\sqsubseteq S \triangleq \{ P \mid \exists Q : P \subseteq Q, Q \in S \}$. We then extend the refinement order to a preorder on sets of pomsets: $S \sqsubseteq S' \triangleq S \sqsubseteq S'$. (This definition is equivalent to $S \subseteq S' \triangleq S \subseteq S'$.)

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 $\Sigma$ is written $\Sigma^+$:

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

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

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

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

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 $\text{biKA}(e, f)$ asks if $[e] \subseteq [f]$.
- **Definition 2.** Given two expressions $e, f$, the problem $\text{CKA}(e, f)$ asks if $[e] \sqsubseteq [f]$.

The first problem, $\text{biKA}(e, f)$, 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, $\text{CKA}(e, f)$, 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 $\mathcal{P}_+(X)$ for the set of non-empty subsets of a set $X$.

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

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

If $t = (P, x, P')$ 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 $t^*$. Transitions labelled with $\tau$ 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 $C' \triangleq (C \setminus \bullet t) \cup t^*$, and we write $C \xrightarrow{t,N} C'$. 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

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

We write $C_0 \xrightarrow{t_1; \ldots; t_n,N} C_n$ in this case. If $C_0 = \{\mathcal{p}_{in}\}$, then the run is initial; if $C_n = \{\mathcal{p}_{fin}\}$, 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 1 Example of labelled safe Petri net.](image)
Figure 2 An accepting run of the Petri net in Figure 1.

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

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 $C_{i-1} \setminus t_i$ that happen again in $C_i \setminus t_i$, are linked with dotted lines.

A Petri net is safe if $(C \setminus \bullet t) \setminus t = \emptyset$ 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 $A(N)$ of a Petri net $N$ is a non-deterministic finite state automaton over the alphabet of transitions of $N$; its states are configurations of $N$ (i.e. subsets of $P$), 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 $C t N C'$. Writing $L(B)$ for the usual word language of an automaton $B$, the transition automaton is defined so that we have

$$L(A(N)) \triangleq \left\{ t_1 \ldots t_n \mid \{p_{in}\} \xrightarrow{t_1 \ldots t_n} N \{p_{fin}\} \right\}.$$

2.4 Language of a Petri net

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

$$i \rightarrow_R j \triangleq i < j \land (\exists p \in t_i^* \cap t_j : \forall k, i < k < j \Rightarrow p \notin t_k).$$

The causality relation $\leq_R$ is the reflexive transitive closure of $\rightarrow_R$. 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 from Figure 2, these are shown in Figure 3.

- The graph-trace $G(R)$ of $R$ is the graph $(\langle[1..n], \rightarrow_R\rangle$.
- The transition-pomset $R$ is the pomset $T(R) \triangleq ([1..n], \leq_R, \lambda_R)$, where $\lambda_R(i) \triangleq \ell(t_i)$.
- The pomset-trace $P(R)$, of $R$ is the restriction of $T(R)$ to the set $\{ i \mid \ell(t_i) \in \Sigma \}$ of visible actions.
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 $\mathcal{N} = \langle P, T, p_{in}, p_{fin} \rangle$ be a safe labelled Petri net, $P = \langle X, \leq, \lambda \rangle$ a pomset, and $R = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \ldots C_{n-1} \xrightarrow{t_n} C_n$ 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 $0 \leq i < n$, $X_i \subseteq X$ and $\rho_i$ is a map from $C_i$ to $\mathcal{P}(\langle X_i \rangle)$;
2. for every $0 \leq i < n$,
   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:
   $$X_{i+1} = X_i \setminus \{x\}; \quad \rho_{i+1}(p) = \begin{cases} \{ y \in X_{i+1} \mid x \leq y \} & \text{if } p = p_1 \\ \rho_i(p) \setminus \{x\} & \text{otherwise.} \end{cases}$$
   b. if $\ell(t_{i+1}) = \tau$, then
   $$X_{i+1} = X_i; \quad \rho_{i+1}(p) = \begin{cases} \bigcup_{q \in \bullet t_{i+1}} \rho_i(q) & \text{if } p \in \bullet_{t_{i+1}} \\ \rho_i(p) & \text{otherwise.} \end{cases}$$

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 $\rho_{i+1}$ 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 $\rho_{i+1}$ 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 $P \subseteq \mathcal{P}(R)$.

Proof. Suppose $P \subseteq \mathcal{P}(R)$ and call $\varphi$ the corresponding bijection from $\mathcal{P}(R)$ to $P$. We define, for $0 \leq i \leq n$ and $p \in C_i$:

$$X_i = \{ \varphi(j) \mid i < j \} \quad \rho_i(p) = \{ x \in X_i \mid \forall j \in \text{vpred}(p, i), \varphi(j) \leq_P x \}.$$

\[\text{This proof uses the notion of locus and some of the notations to be introduced in Section 5.2.}\]
We now prove that the sequence \((\rho_i, X_i)\)\(_{0 \leq i \leq n}\) is an accepting reading of \(P\) along \(R\). Clearly, \(X_0 = X\), \(X_n = \emptyset\), and condition 1 is satisfied. Condition 2b being straightforward to check, we only detail the proof for condition 2a.

If \(t_{i+1}\) is visible, we pick \(x = \varphi(i+1)\). It belongs to \(\rho_i(\bullet t_{i+1})\), because 1) \(i < i+1\) entails \(\varphi(i+1) \in X_i\); 2) for every \(j \in \text{vpred}(\bullet t_{i+1}, i)\), \(j \leq_R i+1\) which means \(\varphi(j) \leq_P \varphi(i+1)\).

Because \(\varphi\) witnesses \(P \subseteq \mathbb{P}(R)\), we know that the labels of \(t_{i+1}\) and \(\varphi(i+1)\) are equal. We also get \(X_{i+1} = \{\varphi(j) \mid i+1 < j\} = \{\varphi(j) \mid i < j\} = X_i \setminus \{\varphi(i+1)\}\). If \(p \notin t_{i+1}\), then \(\text{vpred}(p, i+1) = \text{vpred}(p, i)\) which means that

\[
\rho_i(\varphi(i+1)) = \rho_i(p) \cap X_{i+1} = \rho_i(p) \setminus \{\varphi(i+1)\}.
\]

Finally, for \(\rho_{i+1}(t_{i+1}^\bullet)\), notice that

\[
(\forall j \in \text{vpred}(t_{i+1}^\bullet, i+1), \varphi(j) \leq_P \varphi(k)) \iff \varphi(i+1) \leq_P \varphi(k).
\]

If we have an accepting reading \((\rho_i, X_i)\) of \(P\) along \(R\), let \(\{i_1, \ldots, i_m\}\) be the visible events in \(R\). We define \(\varphi(i_k)\) to be the single \(x\) contained in \(X_{i_k} \setminus X_{i_k-1}\) (i.e. the one from condition 2a). The function \(\varphi\) establishes the required subsumption: it is bijective and if \(i \leq_R j\) then \(\varphi(i) \leq \varphi(j)\).

\(\blacktriangleright\) Corollary 4. The language recognised by \(N\) is \(\subseteq\llbracket N \rrbracket\).

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 \(N(e)\) such that \(\llbracket N(e) \rrbracket = [e]\). 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; a formal description is given in Appendix A.

\[N(0) = \begin{array}{c}
\emptyset
\end{array}\]

\[N(1) = \begin{array}{c}
\emptyset
\end{array}\]

\[N(a) = \begin{array}{c}
\bullet a
\end{array}\]

\[N(e_1 + e_2) = \begin{array}{c}
e_1 + e_2
\end{array}\]

\[N(e_1 || e_2) = \begin{array}{c}
e_1 || e_2
\end{array}\]

\[N(e_1 \cdot e_2) = \begin{array}{c}
e_1 \cdot e_2
\end{array}\]

\[N(e^*) = \begin{array}{c}
e^*
\end{array}\]

This construction yields decidability of biKA in exponential space. Indeed we may build the Petri nets \(N(e)\) and \(N(f)\) 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 \textsc{ExpSpace}-complete problem.

\(\blacktriangleright\) Theorem 5. The problem biKA lies in the class \textsc{ExpSpace}.

\(\blacktriangleright\) Proposition 6. The language recognised by \(N(e)\) is \(\subseteq\llbracket e \rrbracket\).

\textbf{Proof.} By construction we have \(\llbracket N(e) \rrbracket = [e]\). We conclude using Corollary 4. \(\blacktriangleright\)
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 \( \pi \) be a permutation of \([1..n]\). The action of \( \pi \) on \( R \) is defined as \( \pi R \triangleq t_{\pi(1)}; \ldots; t_{\pi(n)} \). The permutation \( \pi \) is compatible with \( R \) if it is order-preserving:

\[
\forall i, j, \; i \leq_R j \Rightarrow \pi(i) \leq \pi(j).
\]

▶Lemma 7. If \( \pi \) is compatible with \( R \), then \( C_0 \xrightarrow{\pi R} C_n \), \( G (R) = G (\pi R) \), 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 \( \pi R \). ▶

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

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 \( \varphi \) 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 \( R_2 \).

Proof. Let \( \varphi \) be the order-preserving bijection from \( \mathbb{P} (R_2) \) to \( \mathbb{P} (R_1) \), and let \( R_2 = t_1; \ldots; t_n \). Because of the definition of an equivalent run, we want to find a permutation \( \pi \) that is compatible with \( R_2 \), such that \( \pi R_2 \) follows \( R_1 \) and is economical. We define \( \pi \) using a total ordering of \([1..n]\), which we build in several steps. First, we define an ordering \( \prec_1 \) on \([1..n] \):

- if \( i \leq_R j \) then \( i \prec_1 j \);
- if \( t_i, t_j \) are visible and \( \varphi(i) < \varphi(j) \) then \( i \prec_1 j \);
To transform

This defines a partial ordering, as we cannot have simultaneously \( i <_{R_2} j \) and \( \varphi(j) < \varphi(i) \): \( i <_{R_2} j \Rightarrow \varphi(i) <_{R_1} \varphi(j) \Rightarrow \varphi(i) < \varphi(j) \).

We extend this ordering further as \( \triangleleft_2 \):
- if \( i \triangleleft_1 j \) then \( i \triangleleft_2 j \);
- if \( t_i \) is visible, \( t_j \) is silent, and \( i \) and \( j \) are independent with respect to \( \triangleleft_1 \), then \( i \triangleleft_2 j \);
- \( \triangleleft_2 \) is closed by transitivity.

Let \( \triangleleft \) be a linear extension of \( \triangleleft_2 \), and let \( \pi \) be the associated permutation, defined inductively as:

\[
\pi^{-1}(0) = \min_{< \cdot \cdot n \cdot} \pi^{-1}(i + 1) = \text{succ}_d \pi^{-1}(i).
\]

Clearly, \( \pi \) is compatible with \( R_2 \), as \( \leq_{R_2} \subseteq \triangleleft \). Thus \( \pi R_2 \) is an accepting run and \( \mathcal{G}(\pi R_2) = \mathcal{G}(R_2) \). Hence \( \mathcal{P}(R_1) \subseteq \mathcal{P}(R_2) = \mathcal{P}(\pi R_2) \), and because \( R_2 \) is series-parallel, \( \pi R_2 \) is also series-parallel.

We now check that \( \pi R_2 \) is economical, by contradiction. Let \( t_i \) be a silent transition in \( \pi R_2 \), \( j_0 = \min \{ k > 0 \mid t_k \text{ visible} \} \), and \( i \not\leq_{\pi R_2} j_0 \). This later condition translates as \( \pi^{-1}(i) \not\leq_{R_2} \pi^{-1}(j_0) \). Then either \( \pi^{-1}(j_0) \leq_{R_2} \pi^{-1}(i) \) or the two are independent. In the first case, we would have \( \pi^{-1}(j_0) \triangleleft_1 \pi^{-1}(i) \), and in the second \( \pi^{-1}(j_0) \triangleleft_2 \pi^{-1}(i) \). In both cases, we get \( \pi^{-1}(j_0) < \pi^{-1}(i) \), thus \( j_0 < i \), which contradicts the definition of \( j_0 \).

**Example 11.** Consider the run \( R_2 \) from Figure 2 and the following run \( R_1 \):

![Diagram of R1](image1)

The pomset of \( R_1 \) is \( \mathcal{P}(R_1) = P_a \parallel P_b \parallel (P_d; P_c) \), and we may check that \( \mathcal{P}(R_1) \subseteq \mathcal{P}(R_2) \).

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 \( R_2 \).

![Diagram of R2](image2)

### 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 \( R = C_0 \overset{\tau}{\rightarrow} C_1 \cdots C_{n-1} \overset{\tau}{\rightarrow} C_n \), \( \mathcal{T}(R) = ([1..n], \leq_R, \lambda) \). A locus denotes a pair \( (p, i) \) where \( 0 \leq i \leq n \) 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 \( \bigcup_{0 \leq i \leq n} C_i \times \{ i \} \). We generate an equivalence relation \( \approx_R \) on loci using the following rule:

\[
p \notin \cdot t_i \Rightarrow \langle p, i \rangle \approx_R \langle p, i - 1 \rangle .
\]

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 \textit{source} of a locus \( \langle p, i \rangle \) is the smallest index of its equivalence class:

\[
\text{src} \langle p, i \rangle \overset{\Delta}{=} \min \{ j \mid \langle p, i \rangle \approx_R \langle p, j \rangle \} .
\]

Now we define a preorder \( \preceq_R \), generated by the following rules:

\[
p, q \in \cdot t_i \times \cdot t_j \Rightarrow \langle p, i \rangle \preceq_R \langle q, j \rangle , \quad \langle p, i \rangle \approx_R \langle q, j \rangle \Rightarrow \langle p, i \rangle \preceq_R \langle q, j \rangle .
\]

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

\[
\text{pred} \langle p, 0 \rangle \overset{\Delta}{=} \emptyset .
\]

\[
\text{if } p \in \cdot t_i , \text{ then } \text{pred} \langle p, i \rangle \overset{\Delta}{=} \{ i \} \cup \bigcup_{p \in \cdot t_i} \text{pred} \langle q, i - 1 \rangle .
\]

\[
\text{if } p \notin \cdot t_i , \text{ then } \text{pred} \langle p, i \rangle \overset{\Delta}{=} \text{pred} \langle p, i - 1 \rangle .
\]

The \textit{set of visible predecessors of a locus}, written \( \text{vpred} \langle p, i \rangle \), is the subset of indices of visible transitions in \( \text{pred} \langle p, i \rangle \)

\[\blacktriangleleft\text{Lemma 12.}\text{ The following properties hold:}\]

\[
\forall i, \forall p \in \cdot t_i , \text{pred} \langle p, i \rangle = \{ j \mid j \leq_R i \} , \quad (1) \\
\forall i, \forall p \in C_i , \text{pred} \langle p, i \rangle = \{ j \mid j \leq_R \text{src} \langle p, i \rangle \} , \quad (2) \\
\forall i, j, p, q, \langle p, i \rangle \approx_R \langle q, j \rangle \Rightarrow \text{pred} \langle p, i \rangle = \text{pred} \langle q, j \rangle , \quad (3) \\
\forall i, j, p, q, \langle p, i \rangle \preceq_R \langle q, j \rangle \Rightarrow \text{pred} \langle p, i \rangle \subseteq \text{pred} \langle q, j \rangle , \quad (4) \\
\forall i, j, p, \forall q \in \cdot t_j , j \in \text{vpred} \langle p, i \rangle \Rightarrow \langle q, j \rangle \preceq_R \langle p, i \rangle , \quad (5) \\
\forall i \leq_R j , \exists p \in C_j : i \in \text{pred} \langle p, j \rangle . \quad (6)
\]

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

5.3 Schedules

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

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

\[
\eta(0) = (0, 0) \text{ and } \eta(N) = (n_1, n_2); \]

\[
\text{if } \eta(k) = (i, j), \text{ then either} \]

1. \( t_{i+1}^{1} \) is a silent transition and \( \eta(k + 1) = (i + 1, j) \), or
2. \( t_{i+1}^{j} \) is a silent transition and \( \eta(k + 1) = (i, j + 1) \), or
3. \( t_{i+1}^{1} \) and \( t_{j+1}^{2} \) are visible, \( \ell(t_{i+1}^{1}) = \ell(t_{j+1}^{2}) \) and \( \eta(k + 1) = (i + 1, j + 1) \).
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 \( \eta(0) = (0, 0) \) and ends with \( \eta(N) = (n_1, n_2) \), every transition in both runs must be fired. This means there is a label-preserving bijection \( \varphi \), called the bijection induced by \( \eta \), 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 \( \prec_k \subseteq C_1 \times C_2 \) when \( \eta(k) = (i, j) \), by induction:

1. \( \prec_0 = C_0^1 \times C_0^2 \),
2. \( \prec_0 = C_0^1 \times C_0^2 \),
3. \( \eta(k) = (i, j) \), then
   
   \[
   p \prec_{k+1} q \triangleq \begin{cases} 
   p \prec_k q, & \exists p' \in \eta^{1}_{i+1}, p' \prec_k q \text{ if } p \notin (t_{i+1})^*, \\
   & \text{otherwise.} 
   \end{cases}
   \]

The schedule \( \eta \) 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 \( \eta(k) = (\varphi(i), i) \), \( (t_{i+1})^* = \{p\} \) and \( (t_{j+1})^* = \{q\} \).

**Example 15.** Recall the runs \( R_2 \) and \( R_1 \) from Example 11. The following sequence is a schedule from \( R_1 \) to \( R_2 \).

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

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

\[
\begin{align*}
R_2 : & 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \\
R_1 : & 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8
\end{align*}
\]

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 \( \eta \) is valid.
Remark. If \( \eta(k) = (i, j) \), \( \eta(k') = (i', j') \), \( (p, i) \approx_{R_1} (q', i') \) and \( (q, j) \approx_{R_2} (q', j') \), then \( p \prec_k q \) if and only if \( p' \prec_{\varphi} q' \).

Lemma 16. If \( \eta(k) = (i, j) \), then \( p \prec_k q \) entails \( \varphi(\text{vpred}(q, j)) \subseteq \text{vpred}(p, i) \).

Proof. We proceed by induction on \( k \). The statement is obvious for \( k = 0 \), as all sets of predecessors are empty at this point. For the inductive step, if \( \eta(k) = (i, j) \), then either:

1. \( t_{i+1}^1 \) is a silent transition, \( \eta(k + 1) = (i + 1, j) \); suppose \( p \prec_{k+1} q \):
   - if \( q \notin \langle t_{i+1}^1 \rangle^\bullet \), then \( p \prec_k q \) and \( \text{vpred}(p, i) = \text{vpred}(p, i + 1) \); we conclude using the induction hypothesis.
   - if \( q \in \langle t_{i+1}^1 \rangle^\bullet \), then there exists \( p' \in \bullet t_{i+1}^1 \) such that \( p' \prec_k q \). By induction we deduce that \( \varphi(\text{vpred}(q, j)) \subseteq \text{vpred}(p', i) \), hence
     \[
     \varphi(\text{vpred}(q, j)) \subseteq \bigcup_{p' \in \bullet t_{i+1}^1} \text{vpred}(p', i) = \text{vpred}(p, i + 1)
     \]

2. or \( t_{j+1}^2 \) is a silent transition, \( \eta(k + 1) = (i, j + 1) \); suppose \( p \prec_{k+1} q \):
   - if \( q \notin \langle t_{j+1}^2 \rangle^\bullet \), then \( p \prec_k q \) and \( \text{vpred}(q, j + 1) = \text{vpred}(p, i + 1) \); we conclude by induction.
   - if \( q \in \langle t_{j+1}^2 \rangle^\bullet \), then for every \( q' \in \bullet t_{j+1}^2 \) we have \( p \prec_k q' \), and thus \( \varphi(\text{vpred}(q', j)) \subseteq \text{vpred}(p, i) \) by induction; we finally get
     \[
     \varphi(\text{vpred}(q, j + 1)) = \varphi \left( \bigcup_{q' \in \bullet t_{j+1}^2} \text{vpred}(q', j) \right) = \bigcup_{q' \in \bullet t_{j+1}^2} \varphi \left( \text{vpred}(q', j) \right) \subseteq \text{vpred}(p, i)
     \]

3. or \( t_{i+1}^1 \) and \( t_{j+1}^2 \) are visible, \( \ell(t_{i+1}^1) = \ell(t_{j+1}^2) \) and \( \eta(k + 1) = (i + 1, j + 1) \); suppose \( t_{i+1}^1 = \{p_0, \{a, \{p_1, \{q_0, \{a, \{q_1, \}} \}} \} \) and \( p \prec_{k+1} q \). There are three cases:
   - either \( (p, q) = (p_1, q_0) \) and \( p_0 \prec_k q_0 \), which means by induction that \( \varphi(\text{vpred}(q_0, j)) \subseteq \text{vpred}(p_0, i) \). Also notice that \( i + 1 \) and \( j + 1 \) are corresponding indices, in the sense that \( \varphi(j + 1) = i + 1 \), thus we have:
     \[
     \varphi(\text{vpred}(q_1, j + 1)) = \varphi(\{j + 1, \} \cup \text{vpred}(q_0, j))
     \]
   - or \( p = p_1 \) and \( p_0 \prec_k q \), in which case we have
     \[
     \varphi(\text{vpred}(q, j + 1)) = \varphi(\text{vpred}(q, j)) \subseteq \text{vpred}(p_0, i) \subseteq \text{vpred}(p_1, i + 1)
     \]
   - or \( p \neq p_1 \) and \( q \neq q_1 \) and we conclude by induction.

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 \( \mathbb{P}(R_1) \subseteq \mathbb{P}(R_2) \).

Proof. The bijection \( \varphi \) induced by \( \eta \) 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 \( k, p, \varphi \) such that \( \eta(k) = (\varphi(j), j) \), \( \langle t_{i+1}^1 \rangle^\bullet = \{p\} \) and \( \langle t_{j+1}^2 \rangle^\bullet = \{q\} \). By Lemma 12.(1) we have \( i \in \text{vpred}(q, j) \). Since \( \eta \) is valid, we have \( p \prec_k q \), and thus \( \varphi(\text{vpred}(q, j)) \subseteq \text{vpred}(p, \varphi(j)) \) by Lemma 16. This means that \( \varphi(i) \in \text{vpred}(p, \varphi(j)) \), 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 \( \eta \) 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:

- \( \eta(0) = (0, 0) \);
- if \( \eta(k) = (i, j) \) then
  - 1. if \( t_{i+1}^k \) is silent, then \( \eta(k + 1) = (i + 1, j) \);
  - 2. if \( t_{i+1}^k \) is visible and \( t_{j+1}^k \) is silent then \( \eta(k + 1) = (i, j + 1) \);
  - 3. if \( t_{i+1}^k \) and \( t_{j+1}^k \) are visible, then \( \eta(k + 1) = (i + 1, j + 1) \).

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

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

\[ \text{Proof.} \quad \text{See Appendix B.} \]

\[ \text{Proposition 19.} \quad \text{If} \quad R_1 \text{and} \quad R_2 \text{are series-parallel, if} \quad P(R_1) \subseteq P(R_2), \quad \text{and if} \quad R_2 \text{is economical and follows} \quad R_1, \quad \text{then the canonical schedule} \quad \eta \text{from} \quad R_1 \text{to} \quad R_2 \text{is valid.} \]

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

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 \( P(R_1) \subseteq P(R_2) \), 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 \( N_1 \) for which there exists a valid schedule to some run in \( N_2 \).

\[ \text{Definition 20.} \quad \text{The composite automaton} \quad N_1 \times N_2 \quad \text{is the nondeterministic finite state automaton with epsilon-transitions} \quad (Q, T, q_0, F) \quad \text{where:} \]

- the alphabet is the set of transitions of \( N_1 \);
- the set of states \( Q \) consists of triples \( \langle C_1, C_2, \prec \rangle \) with \( C_1 \) and \( C_2 \) respectively configurations of \( N_1 \) and \( N_2 \) and \( \prec \subseteq C_1 \times C_2 \);
- the initial state \( q_0 \) is the triple \( \{p^1_{1n}\}, \{p^2_{1n}\}, \{p^1_{2n}, p^2_{2n}\} \};
- final states are those triples of the shape \( \{p^1_{1f}\}, \{p^2_{1f}\}, \ldots \);
- transitions are split into three kinds:
  - 1. if \( t \) is a silent transition of \( N_1 \), \( C_1 \downarrow_{N_1} C'_1 \), 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
    \[ p \prec' q \iff \begin{cases} p \prec q & \text{if } p \notin t^*, \\ \exists p' \in t, p' \prec q & \text{otherwise}. \end{cases} \]
24:14 On Decidability of Concurrent Kleene Algebra

2. if $t$ is a silent transition of $N_2$, $C_2 \xrightarrow{\emptyset} N_2 C_2'$, then from every state $(C_1, C_2, \prec)$ there is an epsilon-transition going to the state $(C_1, C_2', \prec')$ with

$$p \prec' q \iff \begin{cases} \exists q \in \cdot t, p \prec q' & \text{if } q \notin \cdot t', \\ \forall q' \in \cdot t, p \prec q' & \text{otherwise.} \end{cases}$$

3. if $t_1$ and $t_2$ are visible transitions of $N_1$ and $N_2$ with the same label, inputs $p_0$ and $q_0$ and outputs $p_1$ and $q_1$, if $C_1 \xrightarrow{t_1} N_1 C_1'$ and $C_2 \xrightarrow{t_2} N_2 C_2'$, then from every state $(C_1, C_2, \prec)$ such that $p_0 \prec q_0$, there is a transition labelled with $t_1$ going to the state $(C_1', C_2', \prec')$ with

$$p \prec' q \iff \begin{cases} p_0 \prec q \text{ or } q = q_1 & \text{if } p = p_1, \\ p \prec q \text{ and } q \neq q_1 & \text{otherwise.} \end{cases}$$

By definition of this composite automaton, we have

Lemma 21. The language of the automaton $N_1 \prec N_2$ is the set of accepting runs $R_1$ in $N_1$ such that there is an accepting run $R_2$ in $N_2$ 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 $N_1$ and those in $N_2$ are all series-parallel, then $L(A(N_1)) \subseteq L(N_1 \prec N_2)$ if and only if $[N_1] \subseteq [N_2]$.

Proof. Suppose $L(A(N_1)) \subseteq L(N_1 \prec N_2)$ and let $P \in [N_1]$. There exists $R_1 \in L(A(N_1))$ such that $P = \mathbb{P}(R_1)$. By assumption we also have $R_1 \in L(N_1 \prec N_2)$, which means, by Lemma 21, that there is an accepting run $R_2$ in $N_2$ and a valid schedule $\eta$ from $R_1$ to $R_2$. Proposition 17 then tells us that $\mathbb{P}(R_1) \subseteq \mathbb{P}(R_2)$, thus proving $P \in [N_2]$.

Conversely, assume that $[N_1] \subseteq [N_2]$. Let $R_1 \in L(A(N_1))$ be an accepting run in $N_1$. By assumption, there is an accepting run $R_2$ in $N_2$ such that $P = \mathbb{P}(R_1) \subseteq \mathbb{P}(R_2)$. 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 $R_1 \in L(N_1 \prec N_2)$.

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 $N(e)$ and $N(f)$, and then the finite automata $A(N(e))$ and $N(e) \prec N(f)$. 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 $N(e)$ and $N(f)$ are linear in the size of $e$ and $f$, with at most $2n$ and $2m$ places. The automaton $A(N(e))$ uses at most $2^{2n}$ states (recall these are sets of places). The automaton $N(e) \prec N(f)$ uses at most $2^{2n} \times 2^{2n} \times 2^{2n+2m}$ states. Hence testing language equivalence of these two automata will use an amount of space polynomial in $2^{2n}$ and $2^{2n+2m+4n}$, whence the announced result.
For the lower bound, we reduce the problem of universality of regular expressions with shuffle to the containment of downward-closed rational languages. We briefly recall the former.

\textbf{Regular expressions with shuffle} over the alphabet \(\Sigma\) are terms over the syntax \(e, f \in \text{Rat} \bowtie \langle \Sigma \rangle::= e + f \mid e \cdot f \mid e \bowtie f \mid e^* \mid 0 \mid 1 \mid a \ (a \in \Sigma)\).

Given two words \(u\) and \(v\) over \(\Sigma\), the \textit{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.

\[
\begin{align*}
J_0 \bowtie K &:= \emptyset \\
J_1 \bowtie K &:= \{\varepsilon\} \\
J_a \bowtie K &:= \{a\} \\
J[e + f] \bowtie K &:= J[e] \cup J[f] \\
J[e \cdot f] \bowtie K &:= J[e] \cdot J[f] \\
J[e \bowtie f] \bowtie K &:= \bigcup_{u \in J[e], v \in J[f]} u \bowtie v \\
J[e^*] \bowtie K &:= \bigcup_{n \in \mathbb{N}} \{u_1 \ldots u_n \mid \forall i \leq n, u_i \in [e]\}.
\end{align*}
\]

\textbf{Theorem 24} (Mayer and Stockmeyer [16]). \textit{The problem of testing whether the language of a regular expression with shuffle is equal to} \(\Sigma^*\) \textit{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 \(\preceq [u \parallel v]\) is isomorphic to the shuffle product of \(u\) and \(v\).

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

\textbf{Lemma 25.} \textit{For every word} \(w \in \Sigma^*\), \textit{we have} \(w \in J[e]\) \textit{if and only if} \(w\) \textit{seen as a totally ordered pomset is in} \(\preceq [\llbracket e \rrbracket]\).

\textbf{Proof.} By a simple induction on \(e\), using the above observation for the shuffle case. (Each subcase can be found in [6].) \hfill \blacksquare

As a consequence, the language of \(e\) is \(\Sigma^*\) if and only if \(\llbracket \Sigma^* \rrbracket \subseteq \llbracket [e] \rrbracket\). 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.

\textbf{Theorem 26.} \textit{The problem CKA is ExpSpace-complete.}

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

\section{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 \textit{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 \(\tau\)-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 \textit{place} instead of an initial \textit{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

A Construction of rational Petri nets

- \( \mathcal{N}(0) = \{\{\bullet, \circ\}, \emptyset, \bullet, \circ\}; \mathcal{N}(1) = \{\{\bullet\}, \emptyset, \bullet, \circ\}; \mathcal{N}(a) = \{\{\bullet, \circ\}, \{\{\bullet\}, a\}\}; \mathcal{N}(\circ) = \{\bullet, \circ\} \)

- \( \mathcal{N}(e_1 + e_2): \text{let } \mathcal{N}(e_i) = (\mathcal{P}_i, T_i, p_i, q_i) \text{ for } i = 1, 2, \text{ with } \mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset \text{ and } \bullet, \circ \notin \mathcal{P}_1 \cup \mathcal{P}_2. \)

\[ \mathcal{N}(e + f) = (\mathcal{P}_1 \cup \mathcal{P}_2 \cup \{\bullet, \circ\}, T_1 \cup T_2 \cup T, \bullet, \circ); \]

with \( T = \{\{\bullet\}, \tau, \{p_1\}\}, \{\{\bullet\}, \tau, \{p_2\}\}, \{\{q_1\}, \tau, \{\circ\}\}, \{\{q_2\}, \tau, \{\circ\}\}\).  

- \( \mathcal{N}(e_1 \cdot e_2): \text{let } \mathcal{N}(e_i) = (\mathcal{P}_i, T_i, p_i, q_i) \text{ for } i = 1, 2, \text{ with } \mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset \text{ and } \bullet, \circ \notin \mathcal{P}_1 \cup \mathcal{P}_2. \)

\[ \mathcal{N}(e \cdot f) = (\mathcal{P}_1 \cup \mathcal{P}_2, T_1 \cup T_2 \cup \{\{q_1\}, \tau, \{p_2\}\}, q_1, q_2). \]

- \( \mathcal{N}(e_1 \parallel e_2): \text{let } \mathcal{N}(e_i) = (\mathcal{P}_i, T_i, p_i, q_i) \text{ for } i = 1, 2, \text{ with } \mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset \text{ and } \bullet, \circ \notin \mathcal{P}_1 \cup \mathcal{P}_2. \)

\[ \mathcal{N}(e \parallel f) = (\mathcal{P}_1 \cup \mathcal{P}_2 \cup \{\bullet, \circ\}, T_1 \cup T_2 \cup T, \bullet, \circ); \]

with \( T = \{\{\bullet\}, \tau, \{p_1, p_2\}\}, \{\{q_1, q_2\}, \tau, \{\circ\}\}\).  

- \( \mathcal{N}(e^*): \text{let } \mathcal{N}(e) = (\mathcal{P}, T, p, q) \text{ with } \bullet, \circ \notin \mathcal{P}. \mathcal{N}(e^*) = (\mathcal{P} \cup \{\bullet, \circ\}, T \cup T', \bullet, \circ); \text{ with } \)

\[ T' = \{\{\bullet\}, \tau, \{p\}\}, \{\{\bullet\}, \tau, \{\circ\}\}, \{\{q\}, \tau, \{p\}\}, \{\{q\}, \tau, \{\circ\}\}\}. \]

B Proof of Lemma 18

We first prove two preliminary lemmas.

**Lemma 27.** If \( R \) is a series-parallel run, and if in some intermediate configuration \( C_i \) there exists \( q \in C_i \) such that \( k_1, k_2 \in \text{vpred}(q, i) \), then either \( k_1 \leq R k_2 \) or \( k_2 \leq R k_1 \), or:

\[ \forall p \in C_i, k_1 \in \text{vpred}(p, i) \Leftrightarrow k_2 \in \text{vpred}(p, i). \]

**Proof.** Recall that every series-parallel pomset is N-free [19, Theorem 3.1]. This means that for every quadruple of distinct elements \( a, b, c, d \) such that:

\[ a \leq b \quad a \leq c \quad c \leq d \]

then either \( a \leq c \), \( b \leq d \) or \( c \leq b \). Also notice that if \( G(R) \) is series-parallel, then its set of loci forms such a pomset.

Consider \( q \in C_i \) such that \( k_1, k_2 \in \text{vpred}(q, i) \), with \( k_1 \) and \( k_2 \) are incomparable, and \( p \in C_i \) such that \( k_1 \in \text{vpred}(p, i) \). Let us call \( p_1, p_2 \) the places respectively in the outputs of \( t_{k_1} \) and \( t_{k_2} \) (recall that \( k_1 \) and \( k_2 \) are visible, which means they have a singleton output).

We know that \( (p_1, k_1) \sim_R (q, i) \) and \( (p_1, k_1) \sim_R (p, i) \). Because \( k_1 \) and \( k_2 \) are incomparable, so are \( (p_1, k_1) \) and \( (p_2, k_2) \). Being in the same configuration, \( (q, i) \) and \( (p, i) \) must also be incomparable. Using the previous remark, we get that \( (p_2, k_2) \sim_R (p, i) \), thus proving \( k_2 \in \text{vpred}(p, i) \). 

**Lemma 28.** If \( R \) is economical and \( t_i \) is a visible transition, then for all \( p \in C_i \), either

- \( \text{src}(p, i) \) is visible,
- or there exists \( p' \in C_i \) such that \( \text{src}(p', i) \) is visible and \( \text{pred}(p, i) \subseteq \text{pred}(p', i) \).
Proof. Suppose \( t_i \) is a visible transition, and consider \( p \in C_i \), such that \( k = \text{src} \langle p, i \rangle \) is silent. Let us write \( l = \min \{ j > k \mid t_j \text{ visible} \} \). By definition of economy, \( k \leq q \ l \). As \( t_i \) is visible and \( i > k \), we have \( l \leq i \). This means that there exists \( p' \in C_i \) such that \( l \in \text{pred} \langle p', i \rangle \). If \( \text{src} \langle p', i \rangle = l \) then we are done, otherwise we repeat the argument with \( p' \) instead of \( p \), but notice that the source of \( p' \) is strictly larger (for the ordering of natural numbers) than that of \( p \), and still smaller than \( i \), hence this search will eventually terminate. \( \blacktriangleleft \)

Now we can prove Lemma 18. (If \( R_2 \) is series-parallel and economical, then for every \( k \) with \( \eta(k) = \langle i, j \rangle \), if \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p, i \rangle \), then \( p \not\prec_k q \).)

Proof. We proceed by induction on \( k \), the base case being obvious.

1. If \( \ell_{i+1}^1 \) is silent, there are two cases:
   - Either \( p \notin (\ell_{i+1}^2)^* \), and then \( \text{vpred} \langle p, i+1 \rangle = \text{vpred} \langle p, j \rangle \). Therefore the assumption \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p, i+1 \rangle \) yields \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p, i \rangle \). By induction we get \( p \not\prec_q q \), allowing us to check that \( p \not\prec_{k+1} q \).
   - Or we have \( p \in (\ell_{i+1}^1)^* \). Assume \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p, i+1 \rangle \); in order to prove that \( p \not\prec_{k+1} q \) we need to find some \( p' \in (\ell_{i+1}^1)^* \) such that \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p', i \rangle \).

2. If \( \ell_{j+1}^2 \) is silent, there are also cases to consider:
   - Either \( q \notin (\ell_{j+1}^2)^* \), and then \( \text{vpred} \langle q, j+1 \rangle = \text{vpred} \langle q, j \rangle \). The assumption \( \varphi(\text{vpred} \langle q, j+1 \rangle) \subseteq \text{vpred} \langle p, i \rangle \) entails \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p, i \rangle \), and thus \( p \not\prec_k q \) by induction. From this we get \( p \not\prec_{k+1} q \).
   - Or we have \( q \in (\ell_{j+1}^2)^* \) and \( \text{vpred} \langle q, j+1 \rangle = \bigcup_{p' \in (\ell_{j+1}^2)^*} \text{vpred} \langle q', j \rangle \). The assumption \( \varphi(\text{vpred} \langle q, j+1 \rangle) \subseteq \text{vpred} \langle p, i \rangle \) means that \( \forall q' \in (\ell_{j+1}^2)^*, \text{vpred} \langle q', j \rangle \subseteq \text{vpred} \langle p, i \rangle \), so \( \forall q' \in (\ell_{j+1}^2)^*, p' \not\prec_k q \), hence \( p \not\prec_{k+1} q \).

3. Otherwise, let \( p_0, p_1 \) be the input and output of \( \ell_{i+1}^1 \) and \( q_0, q_1 \) be the input and output of \( \ell_{j+1}^2 \). There are again two cases to consider:
If \( p = p_1 \), then \( \text{vpred} \langle p, i + 1 \rangle = \text{vpred} \langle p_0, i \rangle \cup \{i + 1\} \). By assumption we have \( \varphi(\text{vpred} \langle q, j + 1 \rangle) \subseteq \text{vpred} \langle p, i + 1 \rangle \), and thus

- either \( i + 1 = \varphi(j + 1) \in \varphi(\text{vpred} \langle q, j + 1 \rangle) \), in which case \( q \) must be \( q_1 \), thus \( p = p_1 \prec_{k+1} q_1 = q \);
- or \( \varphi(\text{vpred} \langle q, j \rangle) \subseteq \text{vpred} \langle p_0, i \rangle \), which means that \( p_0 \succ_k q \), hence \( p_1 = p \prec_{k+1} q \).

otherwise \( q \) must be different from \( q_1 \), as \( \varphi(j + 1) = i + 1 \notin \text{vpred} \langle p, i + 1 \rangle \). Thus \( \text{vpred} \langle q, j + 1 \rangle = \text{vpred} \langle q, j \rangle \) and \( \text{vpred} \langle p, i + 1 \rangle = \text{vpred} \langle p, i \rangle \), so we conclude by induction. □