On Decidability of Concurrent Kleene Algebra∗†

Paul Brunet1, Damien Pous2, and Georg Struth3

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 \((K, +, \cdot, \parallel, \star, 0, 1)\) that consist of a Kleene algebra \((K, +, \cdot, \star, 0, 1)\) 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

∗ 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 (CoVoCo, 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).
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.

\[
\begin{array}{ccc}
  a & b & \parallel \\
  c & & \\
  d & d & \\
\end{array}
\]

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 \((a \parallel d) \cdot (b \parallel d)\). 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 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 labelled poset is a triple \( (X, \leq, \lambda) \) where \( X \) is a finite carrier set,
\( \leq \) is a partial order on \( X \) and the map \( \lambda : X \rightarrow \Sigma \) labels every element in \( X \) with a letter in
\( \Sigma \). 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 \( \lambda \); 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 \( \emptyset, \emptyset, [] \) \( (\emptyset \) denoting the empty function);

- for \( a \in \Sigma \), the singleton pomset \( P_a \) is \( \{\bullet\}, \{\bullet, \bullet\}, \bullet \mapsto a\);

- for pomsets \( P_1 = (X_1, \leq_1, \lambda_1) \) and \( P_2 = (X_2, \leq_2, \lambda_2) \) 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:

\[
P_1 \parallel P_2 \triangleq (X_1 \cup X_2, \leq_1 \cup \leq_2, \lambda_1 \cup \lambda_2)
\]

- 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 \( P_2 \):

\[
P_1; P_2 \triangleq (X_1 \cup X_2, \leq_1 \cup \leq_2 \cup X_1 \times X_2, \lambda_1 \cup \lambda_2)
\]

Pomset \( P_1 \) refines \( P_2 \), written \( P_1 \sqsubseteq P_2 \), if there exists a bijective morphism \( \varphi : X_2 \rightarrow X_1 \).
By definition, therefore,

\[
\forall x \in X_2, \lambda_1(\varphi(x)) = \lambda_2(x), \text{ i.e., the bijection preserves labels; and}
\]

\[
\forall x, y \in X_2, x \leq_2 y \Rightarrow \varphi(x) \leq_1 \varphi(y), \text{ i.e., the morphism preserves edges in } P_2.
\]

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: \( S \sqsubseteq S' \triangleq S \sqsubseteq \sqsubseteq S' \). (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 \( \Sigma \) is written \( \text{Rat}^\parallel \langle \Sigma \rangle \).

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

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

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

\[
[0] \triangleq \emptyset \quad [e + f] \triangleq [e] \cup [f] \quad [e^*] \triangleq \bigcup_{n \in \mathbb{N}} \{P_1; \ldots; P_n \mid \forall i \leq n, P_i \in [e]\} \quad [a] \triangleq \{P_a\}.
\]
A set of pomsets is called \((\text{series-rational})\) if it is the language of some expression. It is called \((\text{downward-closed rational})\) if it is the downward-closure of a rational language.

Note that due to the structure of expressions, the pomsets we consider are always \((\text{series-parallel})\): they are built from trivial pomsets by using sequential and parallel compositions. Valdes et al. proved that this property is equivalent to \((\text{N-freeness})\) [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.

\begin{itemize}
\item \textbf{Definition 1.} Given two expressions \(e, f\), the problem \(\text{biKA}(e, f)\) asks if \([e] \subseteq [f]\).
\item \textbf{Definition 2.} Given two expressions \(e, f\), the problem \(\text{CKA}(e, f)\) asks if \([e] \sqsubseteq [f]\).
\end{itemize}

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 \(\text{CKA}\) 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 \textit{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 \textit{labelled Petri net} is a tuple \(\mathcal{N} = \langle P, T, p_{in}, p_{fin}\rangle\) where:

\begin{itemize}
\item \(P\) is a finite set of \textit{places};
\item \(T \subseteq \wp_+(P) \times (\Sigma \cup \{\tau\}) \times \wp_+(P)\) is a set of \textit{labelled transitions};
\item \(p_{in} \in P\) is the \textit{initial place};
\item \(p_{fin} \in P\) is the \textit{final place}.
\end{itemize}

If \(t = \langle P, x, P'\rangle\) is a transition, then \(P\) is its \textit{input set}, written \(\ast t\), \(x\) is its \textit{label}, written \(\ell(t)\), and \(P'\) is its \textit{output set}, written \(t^*\). Transitions labelled with \(\tau\) are called \textit{silent}; the others are called \textit{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 \textit{configuration} is a set of places. A transition \(t\) is \textit{enabled} from a configuration \(C\) if \(\ast t \subseteq C\). 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.

\[ C' \triangleq (C \setminus \bullet t) \cup \bullet t, \] 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

\[ C_0 \xrightarrow{t_1}_{\mathcal{N}} C_1 \xrightarrow{t_2}_{\mathcal{N}} \cdots \xrightarrow{t_{n-1}}_{\mathcal{N}} C_{n-1} \xrightarrow{t_n}_{\mathcal{N}} C_n. \]

We write \( C_0 \xrightarrow{t_1; \ldots; t_n}_{\mathcal{N}} C_n \) in this case. If \( C_0 = \{p_{\text{in}}\} \), then the run is \textit{initial}, if \( C_n = \{p_{\text{fin}}\} \), then it is \textit{final}, and if both conditions hold, it is \textit{accepting}. Finally, a configuration \( C \) is \textit{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 \( C_i \setminus \bullet t_i \), are linked with dotted lines.

A Petri net is \textit{safe} if \((C \setminus \bullet t) \cap \bullet t = \emptyset \) 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 \textit{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 \( \{p_{\text{in}}\} \), its only final state is \( \{p_{\text{fin}}\} \), and its transitions are the triples \((C, t, C')\) 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

\[ \mathcal{L}(\mathcal{A}(\mathcal{N})) \triangleq \left\{ t_1 \ldots t_n \mid \{p_{\text{in}}\} \xrightarrow{t_1; \ldots; t_n}_{\mathcal{N}} \{p_{\text{fin}}\} \right\}. \]

### 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 \textit{immediate causality relation} \( \to_R \subseteq [1..n] \times [1..n] \) as

\[ i \to_R j \triangleq i < j \land (\exists p \in \bullet t_i \cap \bullet t_j : \forall k, \ i < k < j \Rightarrow p \notin \bullet t_k). \]

The \textit{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 from Figure 2, these are shown in Figure 3.

- The \textit{graph-trace} \( \mathcal{G}(R) \) of \( R \) is the graph \( ([1..n], \to_R) \).
- The \textit{transition-pomset} \( R \) is the pomset \( \mathcal{T}(R) \triangleq ([1..n], \leq_R, \lambda_R) \), where \( \lambda_R(i) \triangleq \ell(t_i) \).
- The \textit{pomset-trace} \( \mathcal{P}(R) \), of \( R \) is the restriction of \( \mathcal{T}(R) \) to the set \( \{ i \mid \ell(t_i) \in \Sigma \} \) of visible actions.
On Decidability of Concurrent Kleene Algebra

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 \( \mathcal{N} = (P, T, p_{\text{in}}, p_{\text{fin}}) \) 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 \( 0 \leq i \leq n \), \( X_i \subseteq X \) and \( \rho_i \) is a map from \( C_i \) to \( P \) (\( X_i \));
2. for every \( 0 \leq i < n \),
   a. if \( \ell(t_{i+1}) \in \Sigma \), and if \( \rho_0, p_1 \) 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

\[
X_{i+1} = X_i; \quad \rho_{i+1}(p) = \begin{cases} 
\bigcup_{q \in t_{i+1}} \rho_i(q) & \text{if } p \in t_{i+1}^* \\
\rho_i(p) & \text{otherwise.}
\end{cases}
\]

The reading is initial if \( C_0 = \{ p_{\text{in}} \} \) and \( \rho_0(p_{\text{in}}) = X_0 = X \). The reading is final if \( C_n = \{ p_{\text{fin}} \} \) 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.

\textbf{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 \( \rho_0 = [p_{\text{in}} \rightarrow X] \). 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 \( x \in \rho_i(t_{i+1}) \) 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 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 $P \sqsubseteq P(R)$.

Proof. See [3].

Corollary 4. The language recognised by $N$ is $\subseteq [N]$.

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 $N(e)$ such that $[N(e)] = [e]$. 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.

\begin{align*}
N(0) &= \quad \varepsilon \\
N(1) &= \quad e \\
N(a) &= \quad a \\
N(e_1 + e_2) &= \quad e_1 \quad e_2 \\
N(e_1 \parallel e_2) &= \quad e_1 \quad e_2 \\
N(e_1 \cdot e_2) &= \quad e_1 \quad e_2 \\
N(e^*) &= \quad e
\end{align*}

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 ExpSpace-complete problem.

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

Proposition 6. The language recognised by $N(e)$ is $\subseteq [e]$.

Proof. By construction we have $[N(e)] = [e]$. 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 $N_1$ and $N_2$ for this section and the following one. Our goal is to check whether $[N_1] \sqsubseteq [N_2]$, i.e., whether for each run $R_1 \in \mathcal{L}(N_1)$, there exists a corresponding run $R_2 \in \mathcal{L}(N_2)$ such that $P(R_1) \sqsubseteq P(R_2)$.

The first difficulty is that we may have to reorder runs in $N_2$: 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

$$i \leq_R j \Leftrightarrow \pi(i) \leq_R \pi(j).$$

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 $N_1$ and a run $R_2$ in $N_2$ such that $P(R_1) \subseteq P(R_2)$, 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 $N_1$ and $N_2$, respectively. If $P(R_1) \subseteq P(R_2)$ then there exists an economical and series-parallel run $R'_2$ in $N_2$ that follows $R_1$ and is equivalent to $R_2$.

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

$$R_1: \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}$$

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

$$R'_2: \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}$$
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 
\[
R = C_0 \xrightarrow{t_1} C_1 \cdots \xrightarrow{t_{n-1}} C_{n-1} \xrightarrow{t_n} C_n, 
\]
\( T(R) = ([1..n], \preceq_R, \lambda) \). 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:

\[
p \not\in \mathcal{E}_t \Rightarrow \langle p,i \rangle \approx_R \langle p,i - 1 \rangle.
\]

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:

\[
\text{src} \langle p,i \rangle \triangleq \min \{ j \leq i \mid \langle p,i \rangle \approx_R \langle p,j \rangle \}.
\]

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

\[
p,q \in \mathcal{E}_t \times \mathcal{E}_t \Rightarrow \langle p,i - 1 \rangle \preceq_R \langle q,i \rangle, \quad \langle p,i \rangle \approx_R \langle q,j \rangle \Rightarrow \langle p,i \rangle \preceq_R \langle q,j \rangle.
\]

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

\[
\begin{align*}
\text{pred} (p,0) & \triangleq \emptyset, \\
\text{if } p \in \mathcal{E}_t, \text{ then } \text{pred} (p,i) & \triangleq \{i\} \cup \bigcup_{q \in \mathcal{E}_t} \text{pred} (q,i - 1), \\
\text{if } p \not\in \mathcal{E}_t, \text{ then } \text{pred} (p,i) & \triangleq \text{pred} (p,i - 1).
\end{align*}
\]

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

\begin{lemma}
The following properties hold:
\begin{alignat}{2}
\forall i, \forall p \in \mathcal{E}_t, & \quad \text{pred} (p,i) = \{ j \mid j \leq_R i \}, & \quad (1) \\
\forall i, \forall p \in C_i, & \quad \text{pred} (p,i) = \{ j \mid j \leq_R \text{src} (p,i) \}, & \quad (2) \\
\forall i, j, p, q, & \quad \langle p,i \rangle \approx_R \langle q,j \rangle \Rightarrow \text{pred} (p,i) = \text{pred} (q,j), & \quad (3) \\
\forall i, j, p, q, & \quad \langle p,i \rangle \preceq_R \langle q,j \rangle \Rightarrow \text{pred} (p,i) \subseteq \text{pred} (q,j), & \quad (4) \\
\forall i, j, p, q \in \mathcal{E}_t, & \quad j \in \text{vpred} (p,i) \Rightarrow \langle q,j \rangle \preceq_R (p,i), & \quad (5) \\
\forall i \leq j, & \quad \exists p \in C_j : i \in \text{pred} (p,j). & \quad (6)
\end{alignat}
\end{lemma}

Notice that we managed to lift the causality relation \( \leq_R \) to the level of loci: this lemma 
implies that if \( i \neq j \), \( p \in \mathcal{E}_t \) and \( q \in \mathcal{E}_t \), 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 schedule, where we interleave two runs in such 
a way that they synchronise on visible transitions.

\begin{definition}
Let \( R_1 \) and \( R_2 \) be two runs with \( R_i = C'_0 \xrightarrow{t'_1} C'_1 \cdots \xrightarrow{t'_{n_i-1}} C'_{n_i-1} \xrightarrow{t'_n} C'_n \) for 
\( i \in \{1,2\} \). An \( N \)-schedule from \( R_1 \) to \( R_2 \) is a function \( \eta : [0..N] \rightarrow [0..n_1] \times [0..n_2] \) such that 
\( \eta(0) = (0,0) \) and \( \eta(N) = (n_1, n_2) \); 
\( \text{if } \eta(k) = (i,j), \text{ then either} \)
\end{definition}
1. \( t_{i+1}^1 \) is a silent transition and \( \eta(k+1) = \langle i+1, j \rangle \), or
2. \( t_{i+1}^2 \) is a silent transition and \( \eta(k+1) = \langle i, j +1 \rangle \), or
3. \( t_{i+1}^1 \) and \( t_{i+1}^2 \) are visible, \( \ell(t_{i+1}^1) = \ell(t_{i+1}^2) \) and \( \eta(k+1) = \langle i+1, j +1 \rangle \).

Note that a schedule constructs a bijection between the visible transitions of \( R_i \) 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 \( \varphi \), 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 \( \prec_k \subseteq \mathcal{C}_1 \times \mathcal{C}_2 \) when \( \eta(k) = \langle i, j \rangle \), by induction:

\[
\prec_0 = \mathcal{C}_1 \times \mathcal{C}_2; \quad \text{if } \eta(k) = \langle i, j \rangle, \text{ then }
\]

1. if \( \eta(k+1) = \langle i+1, j \rangle \), we set \( \prec_{k+1} q \equiv \begin{cases} p \prec_k q & \text{if } p \notin (t_{i+1}^1) \star, \\ \exists p' \in \cdot t_{i+1}^1 : p' \prec_k q & \text{otherwise}. \end{cases} \)
2. if \( \eta(k+1) = \langle i, j +1 \rangle \), we set \( \prec_{k+1} q \equiv \begin{cases} p \prec_k q & \text{if } q \notin (t_{j+1}^2) \star, \\ \forall q' \in \cdot t_{j+1}^2 : p \prec_k q' & \text{otherwise}. \end{cases} \)
3. otherwise, let \( t_{i+1}^1 = \langle \{p_0\}, a, \{p_1\} \rangle \) and \( t_{j+1}^2 = \langle \{q_0\}, a, \{q_1\} \rangle \); we set
\[
p \prec_{k+1} q \equiv \begin{cases} p_0 \prec_k q & \text{if } (p_1 \prec_k q_1) \text{ or } (p_0 \prec_k q_1) \\ p \prec_k q & \text{if } p = p_1 \\ \text{and } q \neq q_1 & \text{otherwise}. \end{cases}
\]

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 \( \eta(k) = \langle \varphi(i), i \rangle \), \( (t_{\varphi(i)}^1) \star = \{p\} \) and \( (t_{j+1}^2) \star = \{q\} \).

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

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

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 \( i \leq_R j \).

We need to show that \( \varphi(i) \leq_{R_1} \varphi(j) \).

Take the unique \( k, p, q \) such that \( \eta(k) = (\varphi(j), j) \), \( \langle t^i_{i+1}(j) \rangle^* = \{p\} \) and \( \langle t^j_{j+1}(j) \rangle^* = \{q\} \). By Lemma 12.1 we have \( i \in v_{\text{pred}}(q, j) \). Since \( \eta \) is valid, we have \( p \prec_k q \), and thus \( \varphi(v_{\text{pred}}(q, j)) \subseteq v_{\text{pred}}(p, \varphi(j)) \) by Lemma 16. This means that \( \varphi(i) \in v_{\text{pred}}(p, \varphi(j)) \), and using Lemma 12.1 again we obtain \( \varphi(i) \leq_{R_1} \varphi(j) \).

Remark. \( \langle t^i_{i+1}(j) \rangle^* \) is an order preserving bijection from the pomsets of \( R_2 \) to that of \( R_1 \).

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:

\[
\begin{align*}
\eta(0) &= (0, 0); \quad \\
\text{if } \eta(k) &= (i, j) \text{ then } \\
&\quad 1. \text{ if } t^i_{i+1} \text{ is silent, then } \eta(k + 1) = (i + 1, j); \\
&\quad 2. \text{ if } t^i_{i+1} \text{ is visible and } t^j_{j+1} \text{ is silent then } \eta(k + 1) = (i, j + 1); \\
&\quad 3. \text{ if } t^i_{i+1} \text{ and } t^j_{j+1} \text{ are visible, then } \eta(k + 1) = (i + 1, j + 1).
\end{align*}
\]

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.

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

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

Proof. First note that since \( R_2 \) follows \( R_1 \), \( \varphi \) and the bijection witnessing \( \mathbb{P}(R_1) \subseteq \mathbb{P}(R_2) \) must coincide. Let \( i \) be a visible index in \( R_2 \), and let \( k, p, q \) such that \( \eta(k) = (\varphi(i), i) \), \( \langle t^i_{i+1}(i) \rangle^* = \{p\} \) and \( \langle t^j_{j+1}(j) \rangle^* = \{q\} \). We have to prove \( p \prec_k q \). By Lemma 18, it suffices to prove the inclusion \( \varphi(v_{\text{pred}}(q, i)) \subseteq v_{\text{pred}}(p, \varphi(i)) \), i.e., that for every \( j \in v_{\text{pred}}(q, i) \), we have \( \varphi(j) \in v_{\text{pred}}(p, \varphi(i)) \). This is equivalent to checking that for every visible \( j, j \leq_R i \) implies \( \varphi(j) \leq_{R_1} \varphi(i) \), which is true because \( \varphi \) 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 \( \mathbb{P}(R_1) \subseteq \mathbb{P}(R_2) \), 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 \( N_1 \) for which there exists a valid schedule to some run in \( N_2 \).
The composite automaton \( N_1 \prec N_2 \) is the nondeterministic finite state automaton with epsilon-transitions \((Q, T_1, q_0, F)\) where:

- the alphabet is the set of transitions of \( N_1 \);
- the set of states \( Q \) consists of triples \((C_1, C_2, \prec)\) 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 \((\{p_{i0}^1\}, \{p_{i0}^2\}, \{(p_{i0}^1, p_{i0}^2)\})\);
- final states are those triples of the shape \((\{p_{f0}^1\}, \{p_{f0}^2\}, -\));
- transitions are split into three kinds:
  1. if \( t \) is a silent transition of \( N_1 \), \( C_1 \xrightarrow{\epsilon} N_1 \) \( C_1' \), then from every state \((C_1, C_2, \prec)\) there is a transition labelled with \( t \) going to the state \((C_1', C_2, \prec')\) with
    \[
    p \prec' q \Leftrightarrow \begin{cases} 
    p \prec q & \text{if } p \notin t^*, \\
    \exists p' \in \bullet t, p' \prec q & \text{otherwise.}
    \end{cases}
    \]

  2. if \( t \) is a silent transition of \( N_2 \), \( C_2 \xrightarrow{\epsilon} N_2 \) \( C_2' \), then from every state \((C_1, C_2, \prec)\) there is an epsilon-transition going to the state \((C_1, C_2', \prec')\) with
    \[
    p \prec' q \Leftrightarrow \begin{cases} 
    p \prec q & \text{if } q \notin t^*, \\
    \forall q' \in \bullet t, p \prec q' & \text{otherwise.}
    \end{cases}
    \]

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

By definition of this composite automaton, we have

**Lemma 21.** The language of the automaton \( N_1 \prec N_2 \) 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 \( \mathcal{L}(A(N_1)) \subseteq \mathcal{L}(N_1 \prec N_2) \) if and only if \([N_1] \subseteq [N_2]\).

**Proof.** Suppose \( \mathcal{L}(A(N_1)) \subseteq \mathcal{L}(N_1 \prec N_2) \) and let \( P \in \llbracket N_1 \rrbracket \). There exists \( R_1 \in \mathcal{L}(A(N_1)) \) such that \( P = \mathbb{P}(R_1) \). By assumption we also have \( R_1 \in \mathcal{L}(N_1 \prec N_2) \), 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 \( P \in \llbracket N_2 \rrbracket \).

Conversely, assume that \([N_1] \subseteq [N_2]\). Let \( R_1 \in \mathcal{L}(A(N_1)) \) be an accepting run in \( N_1 \). By assumption, there is an accepting run \( R_2 \) in \( N_2 \) such that \( \mathbb{P}(R_1) \subseteq \mathbb{P}(R_2) \). 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 \( R_1 \in \mathcal{L}(N_1 \prec N_2) \).
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 $N(e) \preceq N(f)$. 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 $N(e) \preceq N(f)$ uses at most $2^{2n} \times 2^{2m} \times 2^{2n+2m}$ states. Hence testing language equivalence of these two automata will use an amount of space polynomial in $2^{2n}$ and $2^{2n+2m+4m}$, 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 $\Sigma$ are terms over the syntax

$$e, f \in \text{Rat}^\infty(\Sigma) := e + f \mid e \cdot f \mid e \natural f \mid e^* \mid 0 \mid 1 \mid a \quad (a \in \Sigma).$$

Given two words $u$ and $v$ over $\Sigma$, the shuffle product of $u$ and $v$, written $u \natural v$, is the language of all words of the form $u_1 v_1 u_2 v_2 \cdots u_k v_k$; 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.

$$[0] := \emptyset \quad [1] := \{e\} \quad [a] := \{a\} \quad [e + f] := [e] \cup [f] \quad [e \cdot f] := [e] \cdot [f] \quad [e \natural f] := \bigcup_{u \in [e], v \in [f]} u \natural v \quad [e^*] := \bigcup_{n \in \mathbb{N}} \{u_1 \ldots u_n \mid \forall i \leq n, u_i \in [e]\}.$$

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

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

Lemma 25. For every word $w \in \Sigma^*$, we have $w \in \llbracket e \rrbracket$ if and only if $w$ seen as a totally ordered pomset in $\llbracket \parallel \rrbracket$.\textnormal{\hspace{1em}□}

Proof. By a simple induction on $e$, using the above observation for the shuffle case. (Each subcase can be found in [6].)\textnormal{\hspace{1em}□}

As a consequence, the language of $e$ is $\Sigma^*$ if and only if $\llbracket \Sigma^* \rrbracket \subseteq \llbracket \parallel \rrbracket$. 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.


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

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/.


