2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science

Petri automata for Kleene allegories

Paul Brunet and Damien Pous
Plume team – LIP, CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon, UMR 5668
{paul.brunet,damien.pous}@ens-lyon.fr

***Abstract*—Kleene algebra axioms are complete with respect to both language models and binary relation models. In particular, two regular expressions recognise the same language if and only if they are universally equivalent in the model of binary relations.**

We consider Kleene allegories, i.e., Kleene algebras with two additional operations which are natural in binary relation models: intersection and converse. While regular languages are closed under those operations, the above characterisation breaks. Instead, we give a characterisation in terms of languages of directed and labelled graphs. We then design a finite automata model allowing to recognise such graphs, by taking inspiration from Petri nets.

This model allows us to obtain decidability of identity-free relational Kleene lattices, i.e., the equational theory generated by binary relations on the signature of regular expressions with intersection, but where one forbids unit. This restriction is used to ensure that the corresponding graphs are acyclic. The decidability of graph-language equivalence in the full model remains open.

I. Introduction

We consider binary relations and the operations of union ($\cup$), intersection ($\cap$), composition ($\cdot$), converse ($\cdot^{\smile}$), transitive closure ($\cdot^{+}$), reflexive-transitive closure ($\cdot^{\star}$), and the constants identity (1) and empty relation (0). This model gives rise to an (in)equational theory: a pair of terms e, f made from those operations and some variables $a, b, \ldots$ is a *valid equation*, denoted $\mathrm{Rel} \vDash e = f$, if the corresponding equality holds universally. Similarly, an inequation $\mathrm{Rel} \vDash e \leq f$ is valid when the corresponding containment holds universally. Here are valid equations and inequations: they hold whatever the relations we assign to variables a, b, and c.

$$\mathrm{Rel} \vDash (a \cup b)^\star \cdot b \cdot (a \cup b)^\star = (a^\star \cdot b \cdot a^\star)^+ \qquad (1)$$

$$\mathrm{Rel} \vDash a^\star \leq 1 \cup a \cdot a^{\smile} \cdot a^+ \qquad (2)$$

$$\mathrm{Rel} \vDash a \cdot b \cap c \leq a \cdot (b \cap a^{\smile} \cdot c) \qquad (3)$$

$$\mathrm{Rel} \vDash a^+ \cap 1 \leq (a \cdot a)^+ \qquad (4)$$

Various fragments of this theory have been studied in the literature:

- *Kleene algebra* [7], where one removes intersection and converse, so that terms are plain regular expressions. The equational theory is decidable by Kleene's work [11], and actually PSPACE-complete [14], [15]. Equation (1) lies in this fragment, and one can notice that the two expressions recognise the same language.
- *Kleene algebra with converse*, where one only removes intersection, is also a decidable fragment [3]. It remains PSPACE [5]. Inequation (2) belongs to this fragment.
- (representable, distributive) *allegories* [8], sometimes called positive relation algebras, where transitive and reflexive-transitive closures are not allowed. They are decidable [8, page 208]; Inequation (3) is known as the *modularity law* in this setting.

To the best of our knowledge, the decidability of the whole theory, *Kleene allegories*, is open. Here we obtain several important steps towards the resolution of this problem:

1) we give a characterisation of the full (in)equational theory in terms of graph languages;
2) we design an automata model inspired by Petri nets, that makes it possible to recognise such graphs;
3) we show how to associate such a graph automaton to any term of Kleene allegories;
4) using these graph automata, we give a decision procedure for the fragment where converse and identity are forbidden.

The latter fragment was studied recently by Andréka et al. [1]; its decidability was open as far as we know. The restriction to this fragment allows us to exploit simplifying assumptions about the produced automata, and to obtain a coinductive algorithm for language inclusion (Section V). We actually show that language inclusion for these automata is EXPSPACE-complete (Section VI).

The next problem, which remains open, consists in obtaining the decidability of language inclusion in the full automata model: together with the presented results, this would entail decidability of Kleene allegories. We outline some of the difficulties arising with converse or unit in presence of intersection in Section V-D.

We continue this introductory section by an informal description of the graph language characterisation and our automata model.

A. Languages

In the simple case of Kleene algebra, i.e., without converse and intersection, the (in)equational theory generated by relations can be characterised by using regular languages. Write $[\![e]\!]$ for the language denoted by a regular expression e; for any two regular expressions e, f, we have

$$\mathrm{Rel} \vDash e \leq f \text{ if and only if } [\![e]\!] \subseteq [\![f]\!] \ . \qquad (5)$$

(This result is easy and folklore; proving that this is also equivalent to provability using Kleene algebra axioms [4], [12], [13] is much harder.)

While regular languages are closed under intersection and converse, the above characterisation does not extend to those operations. For intersection, consider two distinct variables a

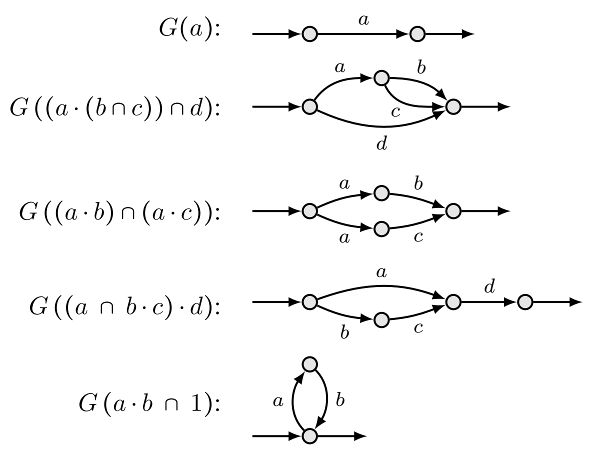

$G(a)$:
a
$G((a\cdot(b\cap c))\cap d)$:
a
b
c
d
$G((a\cdot b)\cap(a\cdot c))$:
a
b
a
c
$G((a\ \cap\ b\cdot c)\cdot d)$:
a
d
b
c
$G(a\cdot b\ \cap\ 1)$:
a
b

Figure 1: Graphs associated to some ground terms.

and b. The extended regular expressions $a\cap b$ and 0 both recognise the empty language, while $\mathrm{Rel}\nvDash a\cap b=0$: one can interpret a and b with intersecting relations. For converse, the extended regular expressions a and $a^{\smile}$ both recognise the singleton language consisting of the single-letter word a. Yet $\mathrm{Rel}\nvDash a=a^{\smile}$, as there are non-symmetric relations.

B. Graphs

Freyd and Scedrov' decision procedure for representable allegories [8, page 208] relies on a notion of directed, labelled, 2-pointed graph. The same notion was proposed independently by Andréka and Bredikhin [2], in a more comprehensive way.

Following Andréka and Bredikhin, we call *ground terms* the terms in the syntax of allegories (composition, intersection, converse, and unit). A ground term u can be represented as a labelled directed graph $G(u)$ with two distinguished vertices called the *input* and the *output*. We give some examples in Figure 1, see Definition 1 for a precise definition.

These graphs can be endowed with a preorder relation: we write $G\blacktriangleleft F$ when there exists a graph homomorphism from F to G preserving labels, inputs, and outputs. For instance the graph corresponding to $(a\cdot(b\cap c))\cap d$ is smaller than the graph of $(a\cdot b)\cap(a\cdot c)$, thanks to the homomorphism depicted in Figure 2 using dotted arrows. Notice that the homomorphism needs not be injective or surjective, so that this preorder has nothing to do with the respective sizes of the graphs: a graph may very well be smaller than another in the sense of $\blacktriangleleft$, while having more vertices or edges (and vice versa).

The key result from Freyd and Scedrov [8, page 208], or Andréka and Bredikhin [2, Theorem 1], is that for any two ground terms u, v, we have

$$\mathrm{Rel}\vDash u\leq v \text{ if and only if } G(u)\blacktriangleleft G(v)\ . \tag{6}$$

The graphs are finite so that one can search exhaustively for a homomorphism, whence the decidability result.

C. Graph languages

To extend the above graph-theoretical characterisation to Kleene allegories, we need to handle union and (reflexive-) transitive closures. It suffices for that to consider sets of graphs: to each expression e, we associate a set of graphs $\mathscr{G}(e)$. This set is most often infinite when the expression e contains (reflexive-)transitive closures.

Writing ${}^{\blacktriangleleft}X$ for the downward closure of a set of graphs X by the preorder $\blacktriangleleft$ on graphs, we obtain the following generalisation of both (5) and (6): for any two expressions e and f,

$$\mathrm{Rel}\vDash e\leq f \text{ if and only if } {}^{\blacktriangleleft}\mathscr{G}(e)\subseteq{}^{\blacktriangleleft}\mathscr{G}(f)\ . \tag{7}$$

This is Theorem 6 in the sequel, and this result is almost there in the work by Andréka et al. [1], [2]. To the best of our knowledge this explicit formulation is new, as well as its use towards decidability results.

When e and f are ground terms, we recover the characterisation (6) for representable allegories: $\mathscr{G}(e)$ and $\mathscr{G}(f)$ are singleton sets in this case. For plain regular expressions, the graphs are just words and the preorder $\blacktriangleleft$ reduces to isomorphism. We thus recover the characterisation (5) for Kleene algebra. This result also generalises the characterisation provided by Ésik et al. [3] for Kleene algebra with converse: graphs of expressions without intersection are just words over a duplicated alphabet, and the corresponding restriction of the preorder $\blacktriangleleft$ precisely corresponds to the word rewriting system they use (see Remark 7).

D. Petri automata

In order to exploit the above characterisation and obtain decidability results, one has first to represent graph languages in a finitary way. We propose for that a new finite automata model, largely based on Petri nets [16]–[18]. We describe this model below, ignoring converse and unit for the sake of clarity.

Recall that a Petri net consists of

- a finite set of *places*, denoted with circles;
- a set of *transitions*, denoted with rectangles;
- for each transition, a set of input places and a set of output places, denoted with arrows;
- an *initial place*, denoted by an entrant arrow;
- a set of *final markings*, denoted by dotted boxes (a marking, or configuration, being a set of places).

The execution model is the following: start by putting a token on the initial place; choose a transition whose input places all contain a token, remove those tokens and put new tokens in the output places of the transition; repeat this process until a final marking is reached. The obtained sequence of transitions is called an *accepting run*. (We actually restrict ourselves to *safe* Petri nets, to ensure that there is always at most one token in a given place when playing this game.)

A *Petri automaton* is just a safe Petri net[1] with variables labelling the outputs of each transition. The automaton depicted

[1]For every reachable marking in a *safe* net, there is at most one token in any given place.

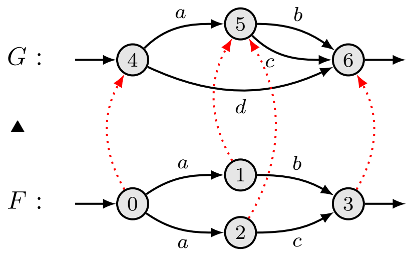
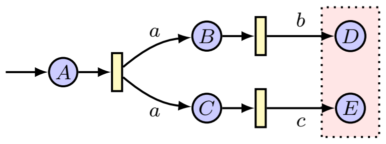


Figure 2: A graph homomorphism.

below is the automaton we will construct for the ground term $a{\cdot}b \cap a{\cdot}c$. Any run must start by firing the left-most transition, reaching the marking $\{B, C\}$; then we have the choice of firing the upper transition first, reaching the marking $\{D, C\}$, or the lower one, reaching the marking $\{B, E\}$. In both cases we reach the final marking $\{D, E\}$ by firing the remaining transition.



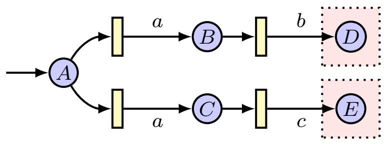
To read a graph in such an automaton, we try to find an accepting run that matches the graph up to homomorphism (Definitions 10 and 11). We do that by using an evolving function from the tokens to the vertices of the graph. We start with the function mapping the unique token, in the initial place, to the input vertex of the graph. To fire a transition, we must check that all its input tokens are mapped to the same vertex in the graph, and that this vertex has several outgoing edges, labelled according to the outputs of the transition. If this is the case, we update the function by removing the mappings corresponding to the deleted tokens, and by adding new mappings for each of the created tokens (using the target vertices of the aforementioned outgoing edges, according to the labels). The graph is accepted by the Petri automaton if we can reach a final marking of the Petri automaton, with all tokens mapped to the output vertex of the graph.

For instance, the previous automaton accepts the graph of $a \cdot b \ \cap \ a \cdot c$ (F in Figure 2). We start with the function $\{A \mapsto 0\}$. We can fire the first transition, updating the function into $\{B \mapsto 1, C \mapsto 2\}$ (We could also choose to update the function into $\{B \mapsto 2, C \mapsto 1\}$, or $\{B, C \mapsto 1\}$, or $\{B, C \mapsto 2\}$ but this would lead to a dead-end). Then we can fire the upper transition, evolving the function into $\{D \mapsto 3, C \mapsto 2\}$, and we finish by firing the remaining transition, obtaining the function $\{D, E \mapsto 3\}$.

We call *language* of $\mathscr{A}$ the set of graphs $\mathscr{L}(\mathscr{A})$ accepted by a Petri automaton $\mathscr{A}$. This language is downward-closed: $\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\mathscr{L}(\mathscr{A})$. For instance, the previous automaton also accepts the graph G from Figure 2, which is smaller than F. Indeed, when we fire the first transition, we can associate the two newly created tokens (in places B and C) to the same vertex (5). This actually corresponds to composing the functions used to accept F with the homomorphism depicted with dotted arrows.

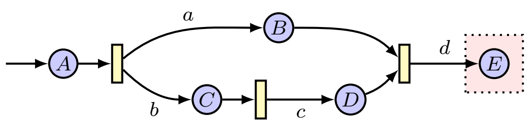
This automata model is expressive enough for Kleene allegories: for any expression e, we can construct a Petri automaton $\mathscr{A}(e)$ such that $\mathscr{L}(\mathscr{A}(e)) = {}^{\blacktriangleleft}\mathscr{G}(e)$ (Section IV). We give three other examples of Petri automata to give more intuition on their behaviour.

The first transition in the previous Petri automaton splits the initial token into two tokens, which are moved concurrently in the remainder of the run. This corresponds to an intersection in the considered expression. This is to be contrasted with the behaviour of the following automaton, which we would construct for the non-ground expression $a{\cdot}b \cup a{\cdot}c$. This automaton has two accepting runs: $\{A\}, \{B\}, \{D\}$ and $\{A\}, \{C\}, \{E\}$, which can be used to accept the (graphs of the) ground terms $a \cdot b$ and $a \cdot c$.

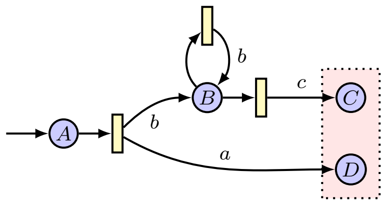


In a sense, two transitions competing for the same tokens represent a non-deterministic choice, i.e., a union in the starting expression.

Still in the first example, the two tokens created by the first transition are later collected in the final marking. Tokens may also be collected and merged by a transition. Consider for instance the following automaton for $(a \cap b{\cdot}c){\cdot}d$. It has only one accepting run, $\{A\}, \{B, C\}, \{B, D\}, \{E\}$, and this run can be used to read the fourth graph from Figure 1.



As a last example, consider the following automaton for the expression $a \ \cap \ b^+ \cdot c$. The upper transition introduces a loop, so that there are infinitely many accepting runs. For any $n > 0$, the graph of the ground term $a \ \cap \ b^n \cdot c$ is accepted by this automaton.



According to characterisation (7), the next step is to decide whether the containment $\mathscr{L}(\mathscr{A}) \subseteq \mathscr{L}(\mathscr{B})$ holds, for two given Petri automata $\mathscr{A}$ and $\mathscr{B}$. Several difficulties arise, that do not appear with classical word automata. Our solution nevertheless uses a standard coinductive approach, where we define an appropriate notion of simulation (Section V-C).

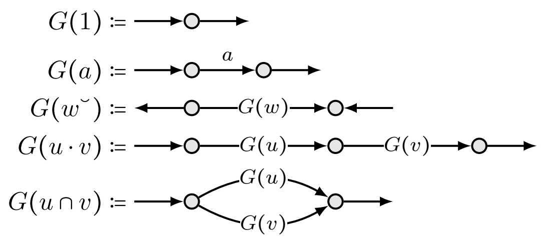


Figure 3: Inductive construction of the graph of a ground term.

Standard notations. For any sets A, B, we write $\mathcal{P}(S) := \{P \mid P \subseteq A\}$ for the set of subsets of A, $A \to B$ for the set of functions from A to B, and $A \rightharpoonup B$ for the set of partial maps from A to B. The domain of a partial map f is denoted by $\mathrm{dom}(f)$.

II. Graph-theoretical characterisation

We consider the signature $\langle \cap, \cup, \cdot, \cdot^{+}, \cdot^{\smile}, 0, 1\rangle$ of Kleene allegories, where $e^{\star}$ is an abbreviation for $1 \cup e^{+}$. We fix a set X of variables, and we denote by $\mathrm{Reg}_X^{\smile\cap}$ the set of expressions $e, f \ldots$ built from variables in X with these connectives. *Ground terms* are the expressions $u, v, w \ldots$ built from X only with the sub-signature $\langle \cap, \cdot, \cdot^{\smile}, 1\rangle$. If $\sigma : X \to \mathcal{P}(S \times S)$ is an interpretation of the alphabet X into some space of relations, we write $\widehat{\sigma}$ for the unique homomorphism extending σ into a function from $\mathrm{Reg}_X^{\smile\cap}$ to $\mathcal{P}(S \times S)$. An inequation between two expressions e and f is *valid*, written $\mathrm{Rel} \vDash e \leq f$, if for any relational interpretation σ we have $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$.

We let G range over 2-pointed labelled directed graphs, which we simply call *graphs* in the sequel. Those are tuples $\langle V, E, \iota, o\rangle$ with V a finite set of vertices, $E \subseteq V \times X \times V$ a set of edges labelled with X, and $\iota, o \in V$ the two distinguished vertices, respectively called *input* and *output*.

Definition 1 (Graph of a ground term: $G(w)$)
To each ground term w, we associate a graph $G(w)$, by induction on w. The graph of $a \in X$ has one edge labelled by a linking its input to its output. The graph for 1 has only one vertex, both input and output. The composition of two graphs with disjoint sets of vertices can be performed by identifying the output of the first graph and the input of the second one. The intersection on graphs consists in merging their inputs and merging their outputs. The converse consists simply in exchanging the input and the output of a graph. $\ast$

See Figure 3 for a graphical description of this construction. Those graphs were introduced independently by Freyd and Scedrov [8, page 208], and Andréka and Bredikhin [2].

Definition 2 (Graph homomorphism, preorders $\blacktriangleleft$ and $\triangleleft$)
A *graph homomorphism* from $\langle V_1, E_1, \iota_1, o_1\rangle$ to $\langle V_2, E_2, \iota_2, o_2\rangle$ is a map $\varphi : V_1 \to V_2$ such that $\varphi(\iota_1) = \iota_2$, $\varphi(o_1) = o_2$, and $(p, x, q) \in E_1$ entails $(\varphi(p), x, \varphi(q)) \in E_2$. We denote by $\blacktriangleleft$ the relation on graphs defined by $G \blacktriangleleft G'$ if there exists a graph homomorphism from G' to G. This relation gives rise to a preorder on ground terms, written $\triangleleft$ and defined by $u \triangleleft v$ if $G(u) \blacktriangleleft G(v)$. $\ast$

Given a set S of graphs, we write ${}^{\blacktriangleleft}S$ for its downward closure: ${}^{\blacktriangleleft}S := \{G \mid G \blacktriangleleft G', G' \in S\}$. Similarly, we write ${}^{\triangleleft}S$ for the downward closure of a set of ground terms w.r.t. $\triangleleft$.

As explained in the introduction, the above preorder on ground terms precisely characterises inclusion under arbitrary relational interpretations:

Theorem 3 ([2, Theorem 1], or [8, page 208])**.** *For all ground terms u, v, we have* $\mathrm{Rel} \vDash u \leq v \Leftrightarrow u \triangleleft v$.

To extend this result to Kleene allegories, we introduce the following generalisation of the language of a regular expression. Sets of words become sets of ground terms.

Definition 4 (Terms and graphs of an expression)
The *set of terms* of an expression $e \in \mathrm{Reg}_X^{\smile\cap}$, written $[\![e]\!]$, is the set of ground terms defined inductively as follows:

$$[\![1]\!] := \{1\} \qquad [\![0]\!] := \varnothing \qquad [\![x]\!] := \{x\}$$
$$[\![e \cdot f]\!] := \{w \cdot w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\}$$
$$[\![e \cap f]\!] := \{w \cap w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\}$$
$$[\![e \cup f]\!] := [\![e]\!] \cup [\![f]\!]$$
$$[\![e^{\star}]\!] := \textstyle\bigcup_{n \in \mathbb{N}} \{w_1 \cdot \cdots \cdot w_n \mid \forall i, w_i \in [\![e]\!]\}$$
$$[\![e^{\smile}]\!] := \{w^{\smile} \mid w \in [\![e]\!]\} \ .$$

The *set of graphs produced* by an expression e, denoted by $\mathscr{G}(e)$ is the set of graphs associated to the ground terms in $[\![e]\!]$:

$$\mathscr{G}(e) := \{G(w) \mid w \in [\![e]\!]\} \ . \qquad \ast$$

To obtain the characterisation announced in the introduction, we need a slight refinement of a lemma established by Andréka, Mikulás, and Németi [1]:

Lemma 5. *For all expression $e \in \mathit{Reg}_X^{\smile\cap} X$, and all relational interpretations $\sigma : X \to \mathcal{P}(S \times S)$, we have*

$$\widehat{\sigma}(e) = \bigcup_{u \in [\![e]\!]} \widehat{\sigma}(u) = \bigcup_{w \in {}^{\triangleleft}[\![e]\!]} \widehat{\sigma}(w) \ .$$

Proof. The first equality is exactly [1, Lemma 2.1]; for the second one, we use the fact that $\widehat{\sigma}(w) \subseteq \widehat{\sigma}(u)$ whenever $w \triangleleft u$, thanks to Theorem 3 (i.e., [2, Theorem 1]). □

Theorem 6. *The following properties are equivalent, for all expressions $e, f \in \mathit{Reg}_X^{\smile\cap}$:*

(i) $\mathrm{Rel} \vDash e \leq f$,
(ii) $[\![e]\!] \subseteq {}^{\triangleleft}[\![f]\!]$,
(iii) $\mathscr{G}(e) \subseteq {}^{\blacktriangleleft}\mathscr{G}(f)$.

Proof. The implication (ii) ⇒ (i) follows easily from Lemma 5, and (iii) ⇒ (ii) is a matter of unfolding definitions. For (i) ⇒ (iii), we mainly use [2, Lemma 3]. □

(The exact characterisation announced in the introduction (7) follows: for any sets X, Y, we have ${}^{\blacktriangleleft}X \subseteq {}^{\blacktriangleleft}Y$ iff $X \subseteq {}^{\blacktriangleleft}Y$.)

Also notice that while $\mathscr{G}(f)$ only contains graphs emanating from ground terms, this is not the case for its closure

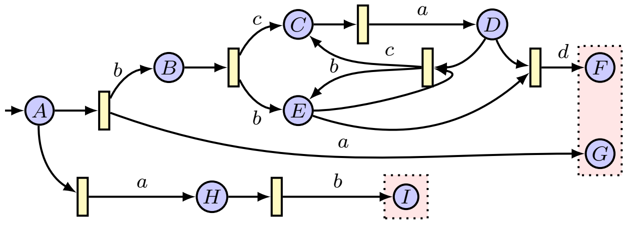
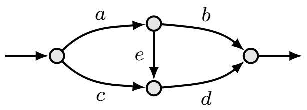


Figure 4: A Petri automaton. The initial place is A, and the final configurations are $\{I\}$ and $\{F,G\}$.

$^{\blacktriangleleft}\mathscr{G}(f)$. For instance, $^{\blacktriangleleft}\mathscr{G}((a\cdot b)\cap(c\cdot d))$ contains the following graph, which is not the graph of any ground term.



Remark 7. The graphs associated to ground terms without intersection are isomorphic to words over a duplicated alphabet. A graph homomorphism between two such graphs is precisely what Ésik et al. call an *admissible map* [3]. Theorem 6 can thus be seen as a generalisation of [3, Theorem 5.3].

III. Petri Automata

We extend the set X of variables into a set $\bar{X}$ of *labels*:$\bar{X} \coloneqq X\cup\{x^{\smile} \mid x\in X\}\cup\{1\}$. A Petri automaton is a Petri net whose transition's outputs are labelled by $\bar{X}$.

Definition 8 (Petri Automaton)
A *Petri automaton* $\mathscr{A}$ over the alphabet X is a tuple $\langle P,\mathscr{T},\iota,\mathscr{F}\rangle$ where:

- P is a finite set of *places*,
- $\mathscr{T}\subseteq\mathcal{P}(P)\times\mathcal{P}\left(\bar{X}\times P\right)$ is a set of *transitions*,
- $\iota\in P$ is the *initial place* of the automaton,
- $\mathscr{F}\subseteq\mathcal{P}(P)$ is a set of *final configurations*, a *configuration* being a set of places.

For each transition $t=\left(\underline{t},\overline{t}\right)\in\mathscr{T}$, $\underline{t}$ and $\overline{t}$ are assumed to be non-empty; $\underline{t}\subseteq P$ is the *input of* t; and $\overline{t}\subseteq\bar{X}\times P$ is the *output of* t. $\star$

We use the graphical notation from the introduction to represent Petri automata; the Petri automaton from Figure 4 will be used as a running example.

From a configuration $\xi\subseteq P$, a transition $t=\left(\underline{t},\overline{t}\right)\in\mathscr{T}$ is *enabled* if $\underline{t}\subseteq\xi$. If so, one may *fire* t, which produces a new configuration $\xi'=\xi\smallsetminus\underline{t}\cup\left\{p\in P \;\middle|\; \exists x\in\bar{X}:(x,p)\in\overline{t}\right\}$. We write $\xi\xrightarrow{t}_{\mathscr{A}}\xi'$ in this case.

A set of transitions $T\subseteq\mathscr{T}$ is *statically compatible* (or just compatible) if their inputs are pairwise disjoint. If furthermore all transitions in T are enabled in a configuration ξ, one can observe that the configuration ξ' reached after firing them successively does not depend on the order in which they are fired. In that case we write $\xi\xrightarrow{T}_{\mathscr{A}}\xi'$.

In the sequel, we assume all considered Petri automata to be *safe*. (I.e., in Petri nets terminology, such that any reachable marking has at most one token in each place [16]). Formally, with our definitions: a Petri automaton $\langle P,\mathscr{T},\iota,\mathscr{F}\rangle$ is safe if for all configuration $\xi\subseteq P$ reachable from $\{\iota\}$ by firing any number of transitions, if $\left(\underline{t},\overline{t}\right)\in\mathscr{T}$ is enabled from ξ, $p\in\xi$, and $(x,p)\in\overline{t}$, then $p\in\underline{t}$.

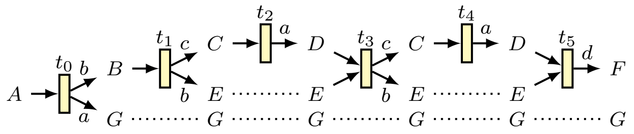


Figure 5: An accepting run in the automaton from Figure 4.

Now we explain how to use Petri automata to define languages of graphs. We first define the *runs* of an automaton.

Definition 9 (Run, accepting run, parallel run)
A *run* is a sequence $\xi=\langle(\xi_k)_{0\leqslant k\leqslant n},(t_k)_{0\leqslant k<n}\rangle$ of configurations and transitions, such that $\xi_k\subseteq P$, $t_k\in\mathscr{T}$ and $\forall k<n,\ \xi_k\xrightarrow{t_k}\xi_{k+1}$. When $\xi_0=\{\iota\}$ and $\xi_n\in\mathscr{F}$, we call ξ an *accepting run*.

A *parallel run* is defined similarly, as a sequence $\Xi=\langle(\Xi_k)_{0\leqslant k\leqslant n},(T_k)_{0\leqslant k<n}\rangle$, where the $T_k\subseteq\mathscr{T}$ are compatible sets of transitions such that $\Xi_k\xrightarrow{T_k}\Xi_{k+1}$. $\star$

(Note that a run ξ is uniquely determined by ξ_0 and the sequence (t_k): all subsequent configurations can be computed deterministically.)

Consider the following sequence of transitions from the automaton in Figure 4:

$$\xi=\langle(\xi_0,\xi_1,\xi_2,\xi_3,\xi_4,\xi_5,\xi_6),(t_0,t_1,t_2,t_3,t_4,t_5)\rangle,$$

with

$$\begin{array}{r|r}
\xi_0=\{A\}, & t_0=(\{A\},\{(b,B),(a,G)\}),\\
\xi_1=\{B,G\}, & t_1=(\{B\},\{(c,C),(b,E)\}),\\
\xi_2=\xi_4=\{C,E,G\}, & t_2=t_4=(\{C\},\{(a,D)\}),\\
\xi_3=\xi_5=\{D,E,G\}, & t_3=(\{D,E\},\{(c,C),(b,E)\}),\\
\xi_6=\{F,G\}. & t_5=(\{D,E\},\{(d,F)\}).
\end{array}$$

We have

$$\{A\}\xrightarrow{t_0}\{B,G\}\xrightarrow{t_1}\{C,E,G\}\underset{t_3}{\overset{t_2,t_4}{\rightleftarrows}}\{D,E,G\}\xrightarrow{t_5}\{F,G\}\quad,$$

and since $\{A\}$ is the initial configuration and $\{F,G\}\in\mathscr{F}$, this sequence is an accepting run. It can be represented graphically as in Figure 5.

As for standard finite state automata, we now need to specify how to read a graph in an automaton. As explained in the introduction, this is done by linking the intermediate configurations of a run to vertices in the graph, and by imposing conditions to match transitions with labelled edges of the graph.

Definition 10 (Reading, parallel reading, language of a run)
A *reading* of $G=\langle V,E,\iota,o\rangle$ along a run $\xi=\langle(\xi_k)_{0\leqslant k\leqslant n},\left(\underline{t_k},\overline{t_k}\right)_{0\leqslant k<n}\rangle$ is a sequence $(\rho_k)_{0\leqslant k\leqslant n}$ such that

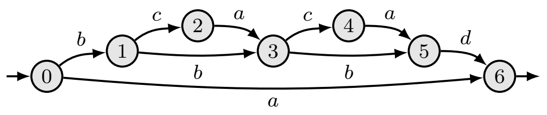


Figure 6: Graph produced by the run depicted in Figure 5.

for all k, ρ_k is a map from ξ_k to V, $\rho_0(\xi_0) = \{\iota\}$, $\rho_n(\xi_n) = \{o\}$, and $\forall k < n$, the following holds:

- all tokens in the input of the transition are mapped to the same vertex in the graph: $\forall p, q \in \underline{t_k},\ \rho_k(p) = \rho_k(q)$;
- the images of tokens in ξ_k that are not in the input of the transition are unchanged: $\forall p \in \xi_k \setminus \underline{t_k},\ \rho_k(p) = \rho_{k+1}(p)$;
- each pair in the output of the transition can be "validated" by the graph: $\forall p \in \underline{t_k}, \forall (x,q) \in \overline{t_k}$,

$$\begin{aligned} x \in X &\Rightarrow (\rho_k(p), x, \rho_{k+1}(q)) \in E, \\ x = y^{\smile} \text{ and } y \in X &\Rightarrow (\rho_{k+1}(q), y, \rho_k(p)) \in E, \\ x = 1 &\Rightarrow \rho_k(p) = \rho_{k+1}(q). \end{aligned}$$

Similarly, we define a *parallel reading* ρ along some parallel run $\Xi = \langle (\Xi_k)_{0 \leqslant k \leqslant n}, (T_k)_{0 \leqslant k < n} \rangle$ by requiring that: $\rho_0(\Xi_0) = \{\iota\}$, $\rho_n(\Xi_n) = \{o\}$, and $\forall k < n$ the following holds:

- $\forall p \in \Xi_k \setminus \bigcup_{(\underline{t},\overline{t}) \in T_k} \underline{t},\ \rho_{k+1}(p) = \rho_k(p)$;
- $\forall (\underline{t}, \overline{t}) \in T_k, \forall p, q \in \underline{t}, \rho_k(p) = \rho_k(q)$;
- $\forall (\underline{t}, \overline{t}) \in T_k, \forall p \in \underline{t}, \forall (x,q) \in \overline{t}$,

$$\begin{aligned} x \in X &\Rightarrow (\rho_k(p), x, \rho_{k+1}(q)) \in E, \\ x = y^{\smile} \text{ and } y \in X &\Rightarrow (\rho_{k+1}(q), y, \rho_k(p)) \in E, \\ x = 1 &\Rightarrow \rho_k(p) = \rho_{k+1}(q). \end{aligned}$$

The *language of a run* ξ, denoted by $\mathscr{L}(\xi)$ is the set of graphs that can be read along ξ. ⋆

The language of a Petri automaton is finally obtained by considering all accepting runs.

Definition 11 (Language recognised by a Petri automaton)
The language recognised by $\mathscr{A}$, written $\mathscr{L}(\mathscr{A})$, is the following set of graphs:

$$\mathscr{L}(\mathscr{A}) \coloneqq \bigcup_{\xi \text{ accepting in } \mathscr{A}} \mathscr{L}(\xi) \quad . \qquad \star$$

The language of a run ξ can be characterised by using a single graph which we call the *graph produced* by ξ: graphs are accepted by ξ exactly when they are smaller than the graph produced by ξ according to $\blacktriangleleft$ (Lemma 14 below). For instance, the run presented in Figure 5 produces the graph depicted in Figure 6.

This graph is obtained in two steps, by first considering a notion of *trace*, which is a graph labelled with $\bar{X}$ rather than X, and which actually corresponds to the notion of pomset-trace from standard Petri nets (see Section VII for more details on this correspondence).

The trace is constructed by creating a vertex k for each transition $t_k = \left(\underline{t_k}, \overline{t_k}\right)$ of the run, plus a final vertex n. We add an edge (k, x, l) whenever there is some place q such that $(x,q) \in \overline{t_k}$, and t_l is the first transition after t_k in the run with q among its inputs, or $l = n$ if there is no such transition in the run.

Definition 12 (Trace of a run)
Let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, \left(\underline{t_k}, \overline{t_k}\right)_{0 \leqslant k < n} \rangle$ be run. For an index $k \leqslant n$ and a place q, let $\nu(k,q)$ be either the smallest index l such that $k \leqslant l$ and $q \in \underline{t_l}$, or n if there is no such index.

The *trace of* ξ is the graph $[\![\xi]\!] \coloneqq \langle \{0, \dots, n\}, E_\xi, 0, n \rangle$ with $E_\xi \coloneqq \left\{ (k, x, \nu(k+1, q)) \;\middle|\; (x,q) \in \overline{t_k} \right\}$. ⋆

To get the final graph, which is labelled by X, one identifies nodes linked by edges labelled by 1, and one replaces each edge of the form $(i, x^{\smile}, j)$ by (j, x, i). Formally:

Definition 13 (Graph produced by a run)
Let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, \left(\underline{t_k}, \overline{t_k}\right)_{0 \leqslant k < n} \rangle$ be run. Let $\equiv_\xi$ be the smallest equivalence relation on $\{0, \dots, n\}$ containing all pairs (i,j) such that $(i, 1, j) \in E_\xi$.

The *graph produced by* ξ, is the graph $\mathscr{G}(\xi)$ defined by

$$\begin{aligned} \mathscr{G}(\xi) &\coloneqq \langle \{[i]_\xi \mid 0 \leqslant i \leqslant n\}, E'_\xi, [0]_\xi, [n]_\xi \rangle \\ [i]_\xi &\coloneqq \{k \in \{0, \dots, n\} \mid i \equiv_\xi k\} \\ E'_\xi &\coloneqq \left\{ ([i]_\xi, x, [j]_\xi) \;\middle|\; \begin{array}{l} x \in X \text{ and} \\ \exists k \in [i]_\xi, l \in [j]_\xi : \\ \quad (k, x, l) \in E_\xi \text{ or} \\ \quad (l, x^{\smile}, k) \in E_\xi \end{array} \right\} \end{aligned} \qquad \star$$

We write $\mathscr{G}(\mathscr{A})$ for the set of graphs produced by accepting runs of a Petri automaton $\mathscr{A}$. To avoid confusions with the language $\mathscr{L}(\mathscr{A})$ of $\mathscr{A}$, we write "G is produced by $\mathscr{A}$" when $G \in \mathscr{G}(\mathscr{A})$, reserving language theoretic terminology like "G is accepted by $\mathscr{A}$" or "$\mathscr{A}$ recognises G" to cases where we mean $G \in \mathscr{L}(\mathscr{A})$.

The graph produced by the run presented in Figure 5 happens to be equal to its trace, since it is labelled in X only. A more involved example is given in Figures 7 to 9. Notice that although the trace of a run is acyclic and can be endowed with a partial order structure (simply check that $\forall p, \nu(\_, p)$ is increasing), it is not necessarily the case for its produced graph.

Lemma 14. *For any accepting run ξ, we have $G \in \mathscr{L}(\xi)$ if and only if $G \blacktriangleleft \mathscr{G}(\xi)$.*

Proof. Suppose there exists a graph homomorphism φ from $\mathscr{G}(\xi)$ to G. Then we can build a reading by defining $\rho_k(p) = \varphi([\nu(k,p)]_\xi)$ for $0 \leqslant k \leqslant n$ and $p \in \xi_k$. On the other hand, if we have a reading $(\rho_k)_{0 \leqslant k \leqslant n}$ of G, we can build a homomorphism φ by letting $\varphi([k]_\xi) = \rho_k(p)$ for any $p \in \underline{t_k}$. As $(\rho_k)_k$ is a reading, φ is well defined. □

As an immediate corollary, we obtain the following characterisation of the language of a Petri automaton.

Corollary 15. $\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\mathscr{G}(\mathscr{A})$.

The left-hand side language is defined through readings along accepting runs, which is a local and incremental notion and which allows us to define *simulations* in Section V-C.

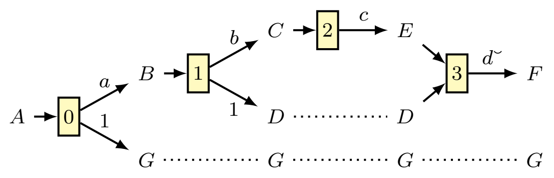


Figure 7: A run ξ.

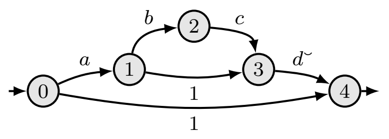


Figure 8: The trace of ξ.

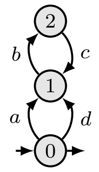


Figure 9: The graph produced by ξ.

By contrast, the right-hand side language is defined globally, which eases the following construction of an automaton recognising the language of an expression.

IV. From expressions to automata

We now show how to associate to any expression $e \in \mathrm{Reg}_X^{\smile\cap}$ an automaton $\mathscr{A}(e)$ that recognises the language ${}^{\blacktriangleleft}\mathscr{G}(e)$. In fact the automaton we obtain has an even stronger connection with e: the graphs in $\mathscr{G}(e)$ are exactly those produced by accepting runs in $\mathscr{A}(e)$. To make the construction simpler, we first modify the expression so that the operator $\cdot^{\smile}$ is only applied to variables, by using the following rewriting system:

$$(a \cup b)^{\smile} \to a^{\smile} \cup b^{\smile} \qquad 0^{\smile} \to 0 \qquad (a^{+})^{\smile} \to (a^{\smile})^{+}$$
$$(a \cdot b)^{\smile} \to b^{\smile} \cdot a^{\smile} \qquad 1^{\smile} \to 1 \qquad a^{\smile\smile} \to a$$
$$(a \cap b)^{\smile} \to a^{\smile} \cap b^{\smile}.$$

(These rules preserve the set of graphs of the expression; also recall that $e^{\star}$ is a shorthand for $e^{+} + 1$, so that we do not need to handle it explicitly.)

The formal construction is inductive; it is given in Definition 16 below. We describe it first in informal terms. For the case of $x \in \bar{X}$, we simply build an automaton with a single transition labelled by x, going from the initial place to a distinct final place[2]. The union consists in putting both automata side by side, and merging their initial places. For the composition of $\mathscr{A}_1$ and $\mathscr{A}_2$ on the other hand we put one automaton in front of the other:

- the initial place of the resulting automaton is that of $\mathscr{A}_1$;
- the final configurations are those of $\mathscr{A}_2$;
- for each final configuration f of $\mathscr{A}_1$, and for each initial transition $(\iota_2, \bar{t})$ coming out of the initial place of $\mathscr{A}_2$, we add a transition with input f and output $\bar{t}$. This last step amounts to adding epsilon transitions from the final configurations of $\mathscr{A}_1$ to the initial place of $\mathscr{A}_2$, and then apply some epsilon-elimination procedure.

[2]Note that this covers uniformly the case of the unit, of variables, and of conversed variables.

We also put the two automata side by side for the intersection, but we merge their initial places, their initial transitions, and their final configurations:

- for any pair of initial transitions of the two automata $(\{\iota_1\}, \overline{t_1}), (\{\iota_2\}, \overline{t_2})$, we put in the intersection automaton the transition $(\{\iota_1\}, \overline{t_1} \cup \overline{t_2})$;
- a final configuration of this automaton is the union of a final configuration from $\mathscr{A}_1$ and a final configuration of $\mathscr{A}_2$.

For the transitive closure $(\cdot^{+})$, we use the ideas for union and composition, adding loops from the final configurations using the initial transitions of the automaton.

Definition 16
To each expression $e \in \mathrm{Reg}_X^{\smile\cap}$, we associate a Petri automaton $\mathscr{A}(e)$ defined inductively as follows:

- $x \in \bar{X}, \mathscr{A}(x) \coloneqq \langle \{0,1\}, \{(\{0\}, \{(x,1)\})\}, 0, \{\{1\}\} \rangle$
- $\mathscr{A}(0) \coloneqq \langle \{0\}, \varnothing, 0, \varnothing \rangle$
- $\mathscr{A}(e_1 \cup e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F}_1 \cup \mathscr{F}_2 \rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \mathscr{T}_2 \cup \{(\{\iota_1\}, \bar{t}) \mid (\{\iota_2\}, \bar{t}) \in \mathscr{T}_2\}$.
- $\mathscr{A}(e_1 \cdot e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F}_2 \rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \mathscr{T}_2 \cup \{(f, \bar{t}) \mid f \in \mathscr{F}_1 \text{ and } (\{\iota_2\}, \bar{t}) \in \mathscr{T}_2\}$.
- $\mathscr{A}(e_1^{+}) \coloneqq \langle P_1, \mathscr{T}, \iota_1, \mathscr{F}_1 \rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \{(f, \bar{t}) \mid f \in \mathscr{F}_1 \text{ and } (\{\iota_1\}, \bar{t}) \in \mathscr{T}_1\}$.
- $\mathscr{A}(e_1 \cap e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F} \rangle$ with

$$\mathscr{F} \coloneqq \{f_1 \cup f_2 \mid f_1 \in \mathscr{F}_1,\ f_2 \in \mathscr{F}_2\} \text{ and}$$
$$\mathscr{T} \coloneqq \{(\underline{t}, \bar{t}) \mid \exists i \in \{1,2\}:\ (\underline{t}, \bar{t}) \in \mathscr{T}_i,\ \iota_i \notin \underline{t}\} \cup \{(\{\iota_1\}, \underline{t_1} \cup \overline{t_2}) \mid \forall i \in \{1,2\},\ (\{\iota_i\}, \overline{t_i}) \in \mathscr{T}_i\}.$$

(In the inductive cases, we assume $\mathscr{A}(e_i) = \langle P_i, \mathscr{T}_i, \iota_i, \mathscr{F}_i \rangle$ for $i \in \{1,2\}$, with $P_1 \cap P_2 = \varnothing$.) $\star$

We prove by induction on e that $\mathscr{A}(e)$ is indeed a safe Petri automaton; for the safety requirement, we add to the induction hypothesis the fact that for any configuration ξ accessible in $\mathscr{A}(e)$, if there is a final configuration $f \in \mathscr{F}$ such that $f \subseteq \xi$, then $f = \xi$. Another invariant is that the initial place never appears in a final configuration, nor in the output of any transition. Note that the place ι_2 becomes unreachable by construction in the cases for union, composition and intersection, so that it could safely be removed, together with the associated transitions.

Theorem 17 (Correctness)**.** *For all expression $e \in Reg_X^{\smile\cap}$, $\mathscr{L}(\mathscr{A}(e)) = {}^{\blacktriangleleft}\mathscr{G}(e)$.*

Proof. We prove a stronger result: $\mathscr{G}(\mathscr{A}(e)) = \mathscr{G}(e)$ (up to graph isomorphisms). This allows us to conclude thanks to Corollary 15. ☐

Corollary 18. *The (in)equational theory of Kleene allegories is co-recursively enumerable.*

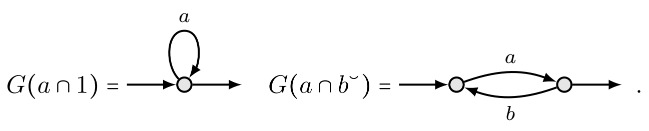
Proof. Construct Petri automata for the two expressions and enumerate all potential counter-examples, i.e., graphs. A graph is a counter-example if it can be read in one automaton but not in the other, which is a decidable property. ☐

Remark 19. If e is an expression without intersection, converse or 1, it can be shown that the transitions in $\mathscr{A}(e)$ are all of the form $(\{p\}, \{(x,q)\})$, with only one input, one output and a label in X. As a consequence, the accessible configurations are singletons, and the resulting Petri automaton has the structure of a non-deterministic finite-state automaton (NFA). Actually, in that case, the construction we described above is just a variation on Thompson's construction [20], with inlined epsilon transition elimination.

V. Comparing automata

A. Simple automata

The above results hold for the whole syntax of regular expressions with converse and intersection. However, in the remainder of the paper, we have to focus on expressions without converse or identity. This is because in combination with intersection, these two operations introduce cycles in the graphs associated to ground terms. Consider for instance the graphs for $a \cap 1$ and $a \cap b^{\smile}$:



Since reflexive-transitive closure $(\cdot^\star)$ implicitly contains an occurrence of the identity, we also have to forbid this operator. Instead, we add the transitive closure $(\cdot^+)$. We thus work with expressions from $\mathrm{Reg}_X^{\cap -}$, defined with the following syntax:

$$e, f \in \mathrm{Reg}_X^{\cap -} ::= x \in X \mid e \cap f \mid e \cup f \mid e \cdot f \mid e^+ \mid 0.$$

Accordingly, ground terms are restricted to the following syntax: $u, v, w ::= x \in X \mid w \cdot w \mid w \cap w$.

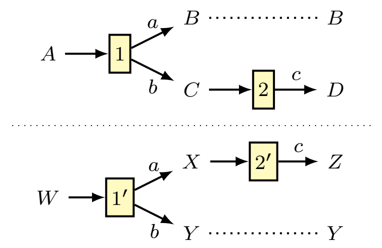
Automata built using Definition 16 from expressions without converse or unit only have transitions labelled with X. This corresponds to the notion of simple automata.

Definition 20 (Simple Petri automaton)
A Petri automaton $\mathscr{A} = \langle P, \mathscr{T}, \iota, \mathscr{F}\rangle$ is called *simple* if $\forall (\underline{t}, \overline{t}) \in \mathscr{T}, \forall (x,p) \in \overline{t}, x \in X$. $\star$

For all $e \in \mathrm{Reg}_X^{\cap -}$, $\mathscr{A}(e)$ is simple. Moreover for any run ξ of a simple Petri automaton, $[\![\xi]\!] = \mathscr{G}(\xi)$ (up to isomorphism); in particular, a simple automaton only produces acyclic graphs.

B. Intuitions

In this section, we show how the notion of simulation relation, that allows to compare NFA, can be adapted to handle simple Petri automata. Consider two automata $\mathscr{A}_1 = \langle P_1, \mathscr{T}_1, \iota_1, \mathscr{F}_1\rangle$ and $\mathscr{A}_2 = \langle P_2, \mathscr{T}_2, \iota_2, \mathscr{F}_2\rangle$, we try to show that for any graph G accepted by $\mathscr{A}_1$, G is recognised by $\mathscr{A}_2$. By Lemma 14, this amounts to proving that for any accepting run ξ in $\mathscr{A}_1$, $\mathscr{G}(\xi)$ is recognised by some accepting run ξ' in $\mathscr{A}_2$. Leaving non-determinism apart, the first idea that comes to mind is to find a relation between the configurations in $\mathscr{A}_1$ and the configurations in $\mathscr{A}_2$, that satisfy some conditions on the initial and final configurations, and such that if $\xi_k \preccurlyeq \xi'_k$ and $\xi_k \xrightarrow{t}_{\mathscr{A}_1} \xi_{k+1}$, then there is a configuration ξ'_{k+1} in $\mathscr{A}_2$ such that $\xi_{k+1} \preccurlyeq \xi'_{k+1}$, $\xi'_k \xrightarrow{t'}_{\mathscr{A}_2} \xi'_{k+1}$, and these transition steps are compatible in some sense. However, such a definition will not give us the result we are looking for. Consider these two runs:



The graphs produced by the first and the second runs correspond respectively to the ground terms $a \cap (b \cdot c)$ and $(a \cdot c) \cap b$. These two terms are incomparable, but the relation $\preccurlyeq$ depicted below satisfies the previously stated conditions.

$$\begin{tikzcd}
\{A\} \arrow[r, "\boxed{1}"] \arrow[d, dash, "\preccurlyeq"] & \{B,C\} \arrow[r, "\boxed{2}"] \arrow[d, dash, "\preccurlyeq"] & \{B,D\} \arrow[d, dash, "\preccurlyeq"] \\
\{W\} \arrow[r, "\boxed{1'}"] & \{X,Y\} \arrow[r, "\boxed{2'}"] & \{Y,Z\}
\end{tikzcd}$$

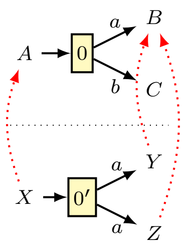
The problem here is that in Petri automata, runs are token firing games. To adequately compare two runs, we need to closely track the tokens. For this reason, we will relate a configuration ξ_k in $\mathscr{A}_1$ not only to a configuration ξ'_k in $\mathscr{A}_2$, but to a map η_k from ξ'_k to ξ_k. This will enable us to associate with each token situated on some place in P_2 another token placed on $\mathscr{A}_1$.

We want to find a reading of $\mathscr{G}(\xi)$ in $\mathscr{A}_2$, i.e., a run in $\mathscr{A}_2$ together with a sequence of maps associating places in $\mathscr{A}_2$ to positions in $\mathscr{G}(\xi)$. Consider the picture below. Since we already have a reading of $\mathscr{G}(\xi)$ along ξ (by defining $\rho_k(p) = \nu(k,p)$, as in the proof of Lemma 14), it suffices to find maps from the places in $\mathscr{A}_2$ to the places in $\mathscr{A}_1$ (the maps η_k): the reading of $\mathscr{G}(\xi)$ in $\mathscr{A}_2$ will be obtained by composing η_k with ρ_k.

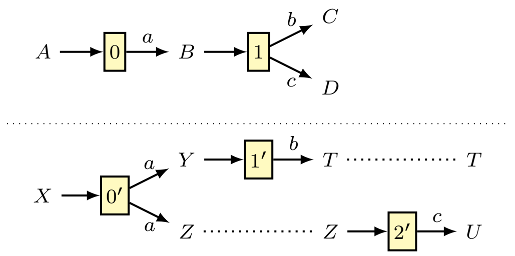
$$\begin{tikzcd}
 & & \mathscr{G}(\xi) & & \\
\xi_0 \arrow[r, "t_0"'] \arrow[urr, "\rho_0"] & \xi_1 \arrow[r, "t_1"'] \arrow[ur, "\rho_1"'] & \cdots \arrow[r, "t_{n-1}"'] & \xi_n \arrow[r, "t_n"'] \arrow[ul, "\rho_n"] & \xi_{n+1} \arrow[ull, "\rho_{n+1}"'] \\
\xi'_0 \arrow[r, "t'_0"'] \arrow[u, "\eta_0"] & \xi'_1 \arrow[r, "t'_1"'] \arrow[u, "\eta_1"] & \cdots \arrow[r, "t'_{n-1}"'] & \xi'_n \arrow[r, "t'_n"'] \arrow[u, "\eta_n"'] & \xi'_{n+1} \arrow[u, "\eta_{n+1}"']
\end{tikzcd}$$

We need to impose some constraints on the maps (η_k) to ensure that $(\rho_k \circ \eta_k)_{0 \leqslant k \leqslant n}$ is indeed a correct reading in $\mathscr{A}_2$. First, we need to ascertain that a transition t'_k in $\mathscr{A}_2$ may be fired from the reading state $\rho_k \circ \eta_k$ to reach the reading state $\rho_{k+1} \circ \eta_{k+1}$. Furthermore, as for NFA, we want transitions t_k and t'_k to be related: specifically, we require t'_k to be included (via the homomorphisms η_k and η_{k+1}) in the transition t_k. This is meaningful because transition t_k contains a lot of information about the vertex k of $\mathscr{G}(\xi)$ and about ρ: the labels of the outgoing edges from k are the labels on the output of t_k, and the only places that will ever be mapped to k in the reading ρ are exactly the places in the input of t_k.

This already shows an important difference between the simulations for NFA and Petri automata. For NFA, we relate a transition $p \overset{a}{\to} p'$ to a transition $q \overset{a}{\to} q'$ with the same label a. Here the transitions $\xi_k \xrightarrow{t_k}_{\mathscr{A}_1} \xi_{k+1}$ and $\xi'_k \xrightarrow{t'_k}_{\mathscr{A}_2} \xi'_{k+1}$ may have different labels. Consider the step represented on the right, corresponding to a square in the above diagram. The output of $\boxed{0}$ has a label b that does not appear in $\boxed{0'}$, and $\boxed{0'}$ has two outputs labelled by a. Nevertheless this satisfies the conditions informally stated above, indeed, $a \cap b \leqslant a \cap a$ holds.



However this definition is not yet satisfactory. Consider the two runs below:



Their produced graphs correspond respectively to the ground terms $a \cdot (b \cap c)$ and $(a \cdot b) \cap (a \cdot c)$. The problem is that $a \cdot (b \cap c) \leqslant (a \cdot b) \cap (a \cdot c)$, but with the previous definition, we cannot relate these runs: they do not have the same length. The solution here consists in grouping the transitions $\boxed{1'}$ and $\boxed{2'}$ together, and consider these two steps as a single step in a *parallel run*. This last modification gives us a notion of simulation that suits our needs.

C. Simulations

Definition 21 (Simulation)
A relation $\preccurlyeq \subseteq \mathcal{P}(P_1) \times \mathcal{P}(P_2 \rightharpoonup P_1)$ between the configurations of $\mathscr{A}_1$ and the partial maps from the places of $\mathscr{A}_2$ to the places of $\mathscr{A}_1$ is called a *simulation* between $\mathscr{A}_1$ and $\mathscr{A}_2$ if:

- if $\xi \preccurlyeq E$ and $\eta \in E$ then the range of η must be included in ξ;
- $\{\iota_1\} \preccurlyeq \{[\iota_2 \mapsto \iota_1]\}$;
- if $\xi \preccurlyeq E$ and $\xi \xrightarrow{(\underline{t},\overline{t})}_{\mathscr{A}_1} \xi'$, then $\xi' \preccurlyeq E'$ where E' is the set of all η' such that there is some $\eta \in E$ and a compatible set of transitions $T \subseteq \mathscr{T}_2$ such that:
  - $\mathrm{dom}(\eta) \xrightarrow{T}_{\mathscr{A}_2} \mathrm{dom}(\eta')$;
  - $\forall (\underline{t'},\overline{t'}) \in T, \eta(\underline{t'}) \subseteq \underline{t}$ and $\forall (x,q) \in \overline{t'}, (x,\eta'(q)) \in \overline{t}$;
  - $\forall p \in \mathrm{dom}(\eta), \left(\forall (\underline{t'},\overline{t'}) \in T, p \notin \underline{t'}\right) \Rightarrow \eta(p) = \eta'(p)$.
- if $\xi \preccurlyeq E$ and $\xi \in \mathscr{F}_1$, then there must be some $\eta \in E$ such that $\mathrm{dom}(\eta) \in \mathscr{F}_2$. $\star$

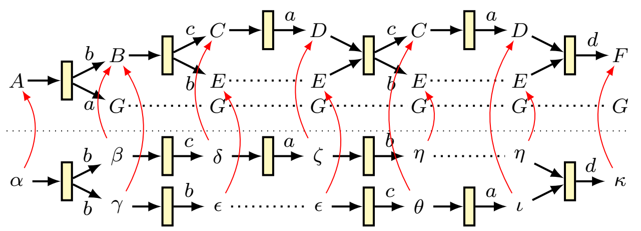


Figure 10: Embedding of a parallel run into the run from Figure 5.

We will now prove that the language of $\mathscr{A}_1$ is contained in the language of $\mathscr{A}_2$ if and only if there exists such a simulation. We first introduce the following notion of embedding.

Definition 22 (Embedding)
Let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, (t_k)_{0 \leqslant i < n} \rangle$ be a run in $\mathscr{A}_1$, and $\Xi = \langle (\Xi_k)_{0 \leqslant k \leqslant n}, (T_i)_{0 \leqslant i < n} \rangle$ a parallel run in $\mathscr{A}_2$. An *embedding* of Ξ into ξ is a sequence $(\eta_i)_{0 \leqslant i \leqslant n}$ of maps such that for any $i < n$, we have:

- η_i is a map from Ξ_i to ξ_i;
- the image of T_i by η_i is included in t_i, meaning that for any $(\underline{t},\overline{t}) \in T_i$, for any $p \in \underline{t}$ and $(x,q) \in \overline{t}$, $\eta_i(p)$ is contained in the input of t_i and $(x, \eta_{i+1}(q))$ is in the output of t_i;
- the image of the tokens in Ξ_i that do not appear in the input of T_i are preserved ($\eta_i(p) = \eta_{i+1}(p)$) and their image is not in the input of t_i.

$$\begin{tikzcd} \xi_i \arrow[r, "t_i"] & \xi_{i+1} \\ \Xi_i \arrow[u, "\eta_i"] \arrow[r, "T_i"'] & \Xi_{i+1} \arrow[u, "\eta_{i+1}"'] \end{tikzcd}$$ $\star$

Figure 10 illustrates the embedding of some parallel run, producing $G(((b \cdot c \cdot a \cdot b) \cap (b \cdot b \cdot c \cdot a)) \cdot d)$, into the run presented in Figure 5. Notice that is it necessary to have a parallel run instead of a simple one: to find something that matches the second transition in the upper run, we need to fire two transitions in parallel in the lower run.

There is a close relationship between simulations and embeddings:

Lemma 23. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two Petri automata, the following are equivalent:*

1) *there exists a simulation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$;*
2) *for any accepting run ξ in $\mathscr{A}_1$, there is an accepting parallel run Ξ in $\mathscr{A}_2$ that can be embedded into ξ.*

Proof. If we have a simulation $\preccurlyeq$, let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, (t_k)_{0 \leqslant k < n} \rangle$ be an accepting run in $\mathscr{A}_1$. By the definition of simulation, we can find a sequence of sets of maps $(E_k)_{0 \leqslant k \leqslant n}$ such that $E_0 = \{[\iota_2 \mapsto \iota_1]\}$ and $\forall k, \xi_k \preccurlyeq E_k$. Furthermore, we can extract from this a sequence of maps $(\eta_k)_{0 \leqslant k \leqslant n}$ and a sequence of parallel transitions $(T_k)_{0 \leqslant k < n}$ such that (η_k) is an embedding of $\langle (\mathrm{dom}(\eta_k))_{0 \leqslant k \leqslant n}, (T_k)_{0 \leqslant k < n} \rangle$ (which is accepting) into ξ.

This follows directly from the definitions of embedding and simulation.

On the other hand, if we have property *2.*, then we can define a relation $\preccurlyeq$ by saying that $\xi \preccurlyeq E$ if there is an accepting run $\xi' = \langle (\xi'_k)_{0\leqslant k\leqslant n}, (t_k)_{0\leqslant k<n}\rangle$ in $\mathscr{A}_1$ such that there is an index k_0: $\xi = \xi'_{k_0}$; and the following holds: $\eta \in E$ if there is an accepting parallel run $\Xi = \langle (\Xi_k)_{0\leqslant k\leqslant n}, (T_k)_{0\leqslant k<n}\rangle$ and $(\eta'_k)_{0\leqslant k\leqslant n}$ an embedding of Ξ into ξ such that $\eta = \eta'_{k_0}$. It is then immediate to check that $\preccurlyeq$ is indeed a simulation. ☐

If η is an embedding of Ξ into ξ, we can easily check that $(\rho_i \circ \eta_i)_{0\leqslant i\leqslant n}$ is a parallel reading of $\mathscr{G}(\xi)$ along Ξ in $\mathscr{A}_2$. Thus, it is clear that once we have such a run Ξ with the sequence of maps η, we have that $\mathscr{G}(\xi)$ is indeed in the language of $\mathscr{A}_2$. The more difficult question is the completeness of this approach: if $\mathscr{G}(\xi)$ is recognised by $\mathscr{A}_2$, is it always the case that we can find a run Ξ that may be embedded into ξ? The answer is affirmative, thanks to Lemma 24 below. If $(\rho_j)_{0\leqslant j\leqslant n}$ is a reading of G along $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (\underline{t_k}, \overline{t_k})_{0\leqslant k<n}\rangle$, we write $active(j)$ for the only position in $\rho_j(\underline{t_j})$[3]. We call $\leqslant$ a *consistent ordering* on $G = \langle V, E, \iota, o\rangle$ if $\langle \overline{V}, \leqslant\rangle$ is a linear order and $(p,x,q) \in E$ entails $p \leqslant q$.

Lemma 24. *Let $G \in \mathscr{L}(\mathscr{A}_2)$ and $\leqslant$ be any consistent ordering on G. Then there exists a run ξ and a reading $(\rho_j)_{0\leqslant j\leqslant n}$ of G along ξ such that $\forall k, active(k) \leqslant active(k+1)$.*

Proof. The proof of this result is achieved by taking any run ξ accepting G, and then exchanging transitions in ξ according to $\leqslant$, while preserving the existence of a reading. The details of this proof being a bit technical, we omit them here. ☐

(Notice that if G contains cycles, this lemma cannot apply because of the lack of consistent ordering.) Lemma 24 enables us to build an embedding from any reading of $\mathscr{G}(\xi)$ in $\mathscr{A}_2$.

Lemma 25. *Let ξ a accepting run of $\mathscr{A}_1$. Then $\mathscr{G}(\xi)$ is in $\mathscr{L}(\mathscr{A}_2)$ if and only if there is an accepting parallel run in $\mathscr{A}_2$ that can be embedded into ξ.*

Proof. We do not include the details of this proof here for length reasons.

For the if direction, we build a parallel reading from the embedding, as explained above. For the other direction, we consider a reading of $\mathscr{G}(\xi)$ in $\mathscr{A}_2$ along some run ξ'. Notice that the natural ordering on $\mathbb{N}$ is consistent for $\mathscr{G}(\xi)$; we may thus change the order of the transitions in ξ' (using Lemma 24) and group them adequately to obtain a parallel reading Ξ that embeds in ξ. ☐

So we know that the existence of embeddings is equivalent to the inclusion of languages, and we previously established that it is also equivalent to the existence of a simulation relation. Hence, the following characterisation holds:

[3]Recall that if $(\rho_j)_{0\leqslant j\leqslant n}$ is a reading along ξ then for any $p, q \in \underline{t_j}$, we have $\rho_j(p) = \rho_j(q)$.

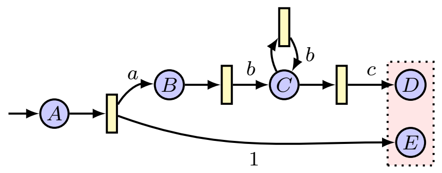


Figure 11: A Petri automaton for $1 \cap a \cdot b^+ \cdot c$.

Theorem 26. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two simple Petri automata. $\mathscr{L}(\mathscr{A}_1) \subseteq \mathscr{L}(\mathscr{A}_2)$ if and only if there exists a simulation relation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$.*

Proof. By Lemmas 14, 23 and 25. ☐

As Petri automata are finite, there are finitely many relations in $\mathcal{P}(\mathcal{P}(P_1) \times \mathcal{P}(P_2 \rightharpoonup P_1))$. The existence of a simulation thus is decidable, allowing us to prove the main result:

Theorem 27. *Given two expressions $e, f \in Reg_X^{\cap -}$, testing whether* $\mathrm{Rel} \vDash e = f$ *is decidable.*

Proof. By Theorems 6, 17 and 26, and reasoning by double inclusion. ☐

In practice, we can build the simulation on-the-fly, starting from the pair $(\{\iota_1\}, \{[\iota_2 \mapsto \iota_1]\})$ and progressing from there. We have implemented this algorithm in OCaml [6]. Even though its theoretical worst case time complexity is huge[4], we get a result almost instantaneously on simple one-line examples.

D. The problems with converse or unit

The previous algorithm is not complete in presence of converse or unit. More precisely, Lemma 25 does not hold for general automata. Indeed, it is not possible to compare two runs just by relating the tokens at each step, and checking each transition independently. Consider the automaton from Figure 11. This automaton has in particular an accepting run recognising $1 \cap abc$. Let us try to test if this is smaller than the following runs from another automaton (we represent the transitions simply as arrows, because they only have a single input and a single output):

$$x_0 \xrightarrow{a} x_1 \xrightarrow{b} x_2 \xrightarrow{c} x_3 \xrightarrow{a} x_4 \xrightarrow{b} x_5 \xrightarrow{c} x_6$$

$$y_0 \xrightarrow{a} y_1 \xrightarrow{b} y_2 \xrightarrow{c} y_3 \xrightarrow{a} y_4 \xrightarrow{b} y_5 \xrightarrow{b} y_6 \xrightarrow{c} y_7$$

It stands to reason that we would reach a point where for the first run:

$$\{D, E\} \preccurlyeq \{[x_3 \mapsto D]\}$$

and for the second run:

$$\{D, E\} \preccurlyeq \{[y_3 \mapsto D]\}.$$

So if it were possible to relate the end of the runs just with this information, they should both be bigger than $1 \cap abc$ or both smaller or incomparable. But in fact the first run

[4]A quick analysis gives a $\mathcal{O}\left(2^{n+(n+1)^m}\right)$ complexity bound, where n and m are the numbers of places of the automata.

(recognising *abcabc*) is bigger than $1 \cap abc$ but the second (recognising *abcabbc*) is not. This highlights the need for having some memory of previously fired transition when trying to compare runs of general Petri automata, thus preventing our local approach to bear fruits. The same kind of example could be found with the converse operation instead of 1.

VI. Complexity

The previous notion of simulation actually allows us to decide language inclusion of simple automata in EXPSPACE. We actually obtain that the problem is EXPSPACE-complete.

Lemma 28. *Comparing simple Petri automata is EXPSPACE-easy.*

Proof. Our measure for the size of an automaton here is its number of places (the number of transitions is at most exponential in this number). Here is a non-deterministic semi-algorithm that tries to refute the existence of a simulation relation between $\mathscr{A}_1$ and $\mathscr{A}_2$.

```
1: start with $\xi \coloneqq \{\iota_1\}$ and $E \coloneqq \{[\iota_2 \mapsto \iota_1]\}$;
2: if $\xi \in \mathscr{F}_1$, check if there is some $\eta \in E$ such that $\mathrm{dom}(\eta) \in \mathscr{F}_2$, if not return FALSE;
3: choose non-deterministically a transition $(\underline{t}, \overline{t}) \in \mathscr{T}_1$ such that $\underline{t} \subseteq \xi$;
4: fire $(\underline{t}, \overline{t})$, which means that
       $$\xi \coloneqq \xi \setminus \underline{t} \cup \{p \in P_1 \mid \exists x \in X : (x,p) \in \overline{t}\};$$
5: have $E$ progress along $(\underline{t}, \overline{t})$ as well, according to the conditions from Definition 21.
6: go to step 2.
```

All these computations can be done in exponential space. In particular as ξ is a set of places in P_1, it can be stored in space $|P_1| \times \log(|P_1|)$. Similarly, E, being a set of partial functions from P_2 to P_1, each of which of size $|P_2| \times \log(|P_1|+1)$, can be stored in space $|P_1+1|^{|P_2|} \times |P_2| \times \log(|P_1|+1)$. This non-deterministic EXPSPACE semi-algorithm can then be turned into an EXPSPACE algorithm by Savitch' theorem [19]. ☐

One can check that the number of places in $\mathscr{A}(e)$ is linear in the size of e. (The exponential upper-bound on the number of transitions is asymptotically reached, consider for instance the automaton for $(x_1 \cup y_1) \cap (x_2 \cup y_2) \cap \cdots \cap (x_n \cup y_n)$.) Therefore, the previous Lemma gives us a EXPSPACE algorithm for deciding the (in)equational theory of identity-free relational Kleene lattices.

Theorem 29. *Comparing simple Petri automata is EXPSPACE-complete.*

Proof. By Lemma 28, it suffices to show hardness. We perform a reduction from the equality of languages denoted by regular expressions with squaring ($e^2 \coloneqq e \cdot e$), an EXPSPACE-complete problem [14]. To avoid confusion, the regular language denoted by the expression e will be written $(\!|e|\!)$. Any word u can be see uniquely as a linear graph $\lambda(u)$. By extension, the set of graphs of words from $(\!|e|\!)$ will be denoted by $\lambda(\!|e|\!)$.

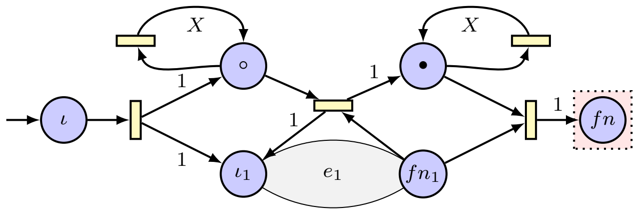


Figure 12: Squaring of the automaton for e_1.

First, notice that if u and v are just words over X, $\lambda(u) \blacktriangleleft \lambda(v)$ is equivalent to $u = v$. Because of this, it is straightforward to check that for any $e, f \in \mathrm{Reg}_X^2$ the following holds

$$^{\blacktriangleleft}(\lambda(\!|e|\!)) = {}^{\blacktriangleleft}(\lambda(\!|f|\!)) \Leftrightarrow (\!|e|\!) = (\!|f|\!) \ . \tag{8}$$

Given an expression e on this signature, we can build in linear time a Petri automaton $\mathscr{A}$, with a linear number of places and transitions. The closure of the language denoted by e is be exactly the language recognised by $\mathscr{A}$. This automaton is not simple: some outgoing arcs are labelled with 1; therefore, this reduction only ensures that the equivalence of arbitrary Petri automata is EXPSPACE-hard. To get EXPSPACE-hardness for simple Petri automata, we need to refine the construction so that 1 is interpreted as a standard letter.

The automata we produce here only have one final configuration, consisting in a singleton. The construction is a straightforward adaptation of Thompson's algorithm for NFA [20]. The only interesting case is for computing an automaton for $e = (e_1)^2$. We represent it graphically in Figure 12. The transitions labelled by X are a shorthand for a set of transitions, containing for each letter x in X a transition with one output, labelled by x. This construction is linear: the automaton for e_1 is not copied. Furthermore, a run in this automaton will start by sending one token in $\circ$ and one in ι_1, the initial state of the automaton for e_1. Then it will perform a run in this automaton until a single token reaches the final state for e_1, fn_1. At this point the tokens from $\circ$ and fn_1 will be sent to $\bullet$ and ι_1, starting a new run of e_1. When a token is finally sent to fn_1, it can be consumed together with the one in $\bullet$, to reach the final configuration. ☐

This proof does not allow us to deduce that also the (in)equational theory of Kleene allegories is EXPSPACE-hard: the Petri automata we construct are not associated to some expressions of polynomial size, a priori.

VII. Relationship with standard Petri net notions

Our notion of Petri automaton is really close to the standard notion of labelled (safe) Petri net, where the transitions themselves are labelled, rather than their outputs. We motivate this design choice, and we relate some of the notions we introduced to the standard ones [16].

Any Petri automaton can be translated into a safe Petri net whose transitions are labelled by $\bar{X} \uplus \{\tau\}$, the additional label τ standing for silent actions. For each automaton transition

$(\{p_1,\ldots,p_n\},\{(x_1,q_1),\ldots,(x_m,q_m)\})$ with $m > 1$, we introduce m fresh places $r_1,\ldots,r_m$ and $m+1$ transitions:

- a silent transition t_0 with preset $\{p_1,\ldots,p_n\}$ and postset $\{r_1,\ldots,r_m\}$;
- and for each $1 \leqslant k \leqslant m$ a transition t_k labelled by x_k, with preset $\{r_k\}$ and postset $\{q_k\}$.

The inductive construction from Section IV is actually simpler to write using labelled Petri nets, as one can freely use τ-labelled transitions to assemble automata into larger ones, one does not need to perform the τ-elimination steps on the fly.

On the other side, we could not define an appropriate notion of simulation for Petri nets: we need to fire several transitions at once in the small net, to provide enough information for the larger net to answer; delimiting which transitions to group and which to separate is non-trivial; similarly, defining a notion of parallel step is delicate in presence of τ-transitions. By switching to our notion of Petri automata, we impose strong constraints about how those τ-transitions should be used, resulting in a more fitted model.

To describe a run in a Petri net N, one may use a *process* $p : K \to N$, where K is an *occurrence net* (a partially ordered Petri net) [9]. The graphical representation (Figures 5, 7 and 10) we used to describe runs in an automaton are in fact a mere adaptation of this notion to our setting (with labels on arcs rather than on transitions).

Our notion $[\![\xi]\!]$ of trace of a run actually corresponds to the standard notion of *pomset-trace*, via dualisation. Jategaonkar and Meyer showed that the pomset-trace equivalence problem for safe Petri nets is EXPSPACE-complete [10]. However this equivalence is too strong and does not coincide with the one discussed in the present paper, even for simple Petri automata. The graph produced by a run is its trace in this case, so that pomset-trace equivalence for Petri nets corresponds to equivalence of the sets of graphs produced by Petri automata ($\mathscr{G}(\mathscr{A}) = \mathscr{G}(\mathscr{B})$, up to graph isomorphism). However, for the equational theory we consider, we need to compare the languages, which are downward-closed sets of graphs, (${}^{\blacktriangleleft}\mathscr{G}(\mathscr{A}) = {}^{\blacktriangleleft}\mathscr{G}(\mathscr{B})$, i.e., $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B})$) rather than the sets of graphs themselves.

Also note that the class of sets of graphs produced by Petri automata $(\{\mathscr{G}(\mathscr{A}) \mid \mathscr{A}$ a Petri automaton$\})$ is not closed under downward closure. Intuitively, the width of any graph in $\mathscr{G}(\mathscr{A})$ is bounded by the number number of places of $\mathscr{A}$, but ${}^{\blacktriangleleft}\mathscr{G}(\mathscr{A})$ usually contains graphs of arbitrary width. As a consequence, one cannot easily reduce our problem to pomset-trace equivalence of safe Petri nets.

VIII. Directions for future work

The automata model we introduced to recognise downward-closed graph languages allowed us to obtain the decidability of identity-free relational Kleene lattices, an (in)equational theory studied recently by Andréka et al. [1]. Thanks to this model, we also obtained that the (in)equational theory of Kleene allegories is co-recursively enumerable. We leave several questions open.

First, is the (in)equational theory of Kleene allegories decidable? Two approaches could provide an affirmative answer: finding an algorithm for comparing arbitrary safe Petri automata, or finding a complete and recursively enumerable axiomatisation.

Second, can we obtain a Kleene-like theorem for Petri automata: is the language of any Petri automaton also the language of a Kleene allegory term? This question can be restricted to simple Petri automata; if one of the answers is negative, is there an algebraic way of representing these automata? Which are the missing operators?

References

[1] H. Andréka, S. Mikulás, and I. Németi. The equational theory of Kleene lattices. *Theoretical Computer Science*, 412(52):7099–7108, 2011.

[2] H. Andréka and D. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995.

[3] S. L. Bloom, Z. Ésik, and G. Stefanescu. Notes on equational theories of relations. *Algebra Universalis*, 33(1):98–126, 1995.

[4] M. Boffa. Une condition impliquant toutes les identités rationnelles. *Informatique Théorique et Applications*, 29(6):515–518, 1995.

[5] P. Brunet and D. Pous. Kleene algebra with converse. In *Proc. RAMiCS*, volume 8428 of *Lecture Notes in Computer Science*, pages 101–118. Springer Verlag, 2014.

[6] P. Brunet and D. Pous. Web appendix to this abstract, 2014. http://perso.ens-lyon.fr/paul.brunet/rklm.html.

[7] J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall Mathematics Series, 1971.

[8] P. J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.

[9] U. Goltz and W. Reisig. The non-sequential behaviour of petri nets. *Information and Control*, 57(2):125–147, 1983.

[10] L. Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107 – 143, 1996. Twentieth International Colloquium on Automata, Languages and Programming.

[11] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. Memorandum. Rand Corporation, 1951.

[12] D. Kozen. A completeness theorem for Kleene Algebras and the algebra of regular events. In *Proc. LICS*, pages 214–225. IEEE Computer Society, 1991.

[13] D. Krob. A Complete System of B-Rational Identities. In *Proc. ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 60–73. Springer Verlag, 1990.

[14] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Switching and Automata Theory, 1972., IEEE Conference Record of 13th Annual Symposium on*, pages 125–129. IEEE, 1972.

[15] A. Meyer and L. J. Stockmeyer. Word problems requiring exponential time. In *Proc. ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.

[16] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989.

[17] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[18] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Darmstadt Univ. of Tech., 1962.

[19] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[20] K. Thompson. Regular expression search algorithm. *C. of the ACM*, 11:419–422, 1968.