Completeness for Identity-free Kleene Lattices

Amina Doumane
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
amina.doumane@ens-lyon.fr

Damien Pous
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
damien.pous@ens-lyon.fr

Abstract

We provide a finite set of axioms for identity-free Kleene lattices, which we prove sound and complete for the equational theory of their relational models. Our proof builds on the completeness theorem for Kleene algebra, and on a novel automata construction that makes it possible to extract axiomatic proofs using a Kleene-like algorithm.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Kleene algebra, Graph languages, Petri Automata, Kleene theorem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.18

Related Version Long version at [13], `https://hal.archives-ouvertes.fr/hal-01780845`.

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157). This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

Relation algebra is an efficient tool to reason about imperative programs. In this approach, the bigstep semantics of a program P is a binary relation $[P]$ between memory states [20, 21, 6, 16, 1]. This relation is built from the elementary relations corresponding to the atomic instructions of P, which are combined using standard operations on relations, for instance composition and transitive closure, that respectively encode sequential composition of programs, and iteration (while loops). Abstracting over the concrete behaviour of atomic instructions, one can compare two programs P, Q by checking whether the expressions $[P]$ and $[Q]$ are equivalent in the model of binary relations, which we write as $\mathcal{R}el \models [P] = [Q]$.

To enable such an approach, one should obtain two properties: decidability of the predicate $\mathcal{R}el \models e = f$, given two expressions e and f as input, and axiomatisability of this relation. Decidability makes it possible to automate the verification process, thus alleviating the burden for the end-user [17, 14, 9, 25, 28]. Axiomatisation offers a better way of understanding the equational theory of relations and provides a certificate for programs verification. Indeed, an axiomatic proof of $e = f$ can be seen as a certificate, which can be exchanged, proofread, and combined in a modular way. Axiomatisations also make it possible to solve hard instances manually, when the existing decision procedures have high complexity and/or when considered instances are large [22, 17, 7].

Depending on the class of programs under consideration, several sets of operations on relations can be considered. In this paper we focus on the following set of operations: composition ($\cdot$), transitive closure ($\_^+$), union ($+$), intersection ($\cap$) and the empty relation (0). The expressions generated by this signature are called KL^--expressions. An example of an inequality in the corresponding theory is $\mathcal{R}el \models (a \cap c) \cdot (b \cap d) \leq (a \cdot b)^+ \cap (c \cdot d)$: when a, b, c, d are interpreted as arbitrary binary relations, we have $(a \cap c) \cdot (b \cap d) \subseteq (a \cdot b)^+ \cap (c \cdot d)$. The operations of composition, union and transitive closure arise naturally when defining the bigstep semantics of sequential programs. In contrast, intersection, which is the operation of interest in the present paper, is not a standard operation on programs. This operation is however useful when it comes to specifications: it allows one to express local conjunctions of specifications. For instance, a specification of the shape $(a \cap b)^+$ expresses the fact that execution traces must consist of sequences of smaller traces satisfying both a and b.

The operations of KL^- contain those of identity-free regular expressions, whose equational theory inherits the good properties of *Kleene algebra* (KA).We summarise them below.

First recall that each regular expression e can be associated with a set of words $\mathcal{L}(e)$ called its language. Valid inequations between regular expressions inequalities can be characterised by language inclusions [29]:

$$\mathcal{R}el \models e \leq f \quad \text{iff} \quad \mathcal{L}(e) \subseteq \mathcal{L}(f) \tag{1}$$

Second, we have the celebrated equivalence between regular expressions and non-deterministic finite automata (NFA) via a *Kleene theorem*: for every regular expression e, there is an NFA such that $\mathcal{L}(e)$ is the language of A, and conversely. Decidability follows (in fact, PSpace-completeness). Lastly, although every purely equational axiomatisation of this theory must be infinite [30], Kozen has proved that Conway's finite quasi-equational axiomatisation [12] is sound and complete [19]. (There is also an independent proof of this result by Boffa [8], based on the extensive work of Krob [26].)

Those three results nicely restrict to identity-free Kleene algebra (KA^-), which form a proper fragment of Kleene algebra [23]. It suffices to consider languages of non-empty words: Equation (1) remains, Kleene's theorem still holds, and we have the following characterisation, where we write $\mathsf{KA}^- \vdash e \leq f$ when $e \leq f$ is derivable from the axioms of KA^-:

$$\mathcal{L}(e) \subseteq \mathcal{L}(f) \quad \text{iff} \quad \mathsf{KA}^- \vdash e \leq f \tag{2}$$

There are counterparts to the first two points for KL^--expressions. Each KL^--expression e can be associated with a set of graphs $\mathcal{G}(e)$ called its graph language, and valid inequations of KL^--expressions can be characterised through these languages of graphs. A subtlety here is that we have to consider graphs modulo homomorphisms; writing ${}^{\triangleleft}\mathcal{G}$ for the closure of a set of graphs $\mathcal{G}$ under graph homomorphisms, we have [10]:

$$\mathcal{R}el \models e \leq f \quad \text{iff} \quad {}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f) \tag{3}$$

KL^--expressions are equivalent to a model of automata over graphs called Petri automata [10]. As for KA^--expressions, a Kleene-like theorem holds [11]: for every KL^--expression e, there is a Petri automaton whose language is $\mathcal{G}(e)$, and conversely. Decidability (in fact, ExpSpace-completeness) of the equational theory follows [10, 11].

What is missing to this picture is an axiomatisation of the corresponding equational theory. In the present paper, we provide such an axiomatisation, which we call KL^-, and which comprises the axioms for identity-free Kleene algebra (KA^-) and the axioms of *distributive lattices* for $\{+, \cap\}$. Completeness of this axiomatisation is the difficult result we prove here:

$${}^{\triangleleft}\mathcal{G}\,(e) \subseteq {}^{\triangleleft}\mathcal{G}\,(f) \quad \text{entails} \quad \mathsf{KL}^- \vdash e \leq f \tag{4}$$

We proceed in two main steps. First we show that $\mathcal{G}(e) \subseteq \mathcal{G}(f)$ entails $\mathsf{KL}^- \vdash e \leq f$, using a technique inspired from [24], this is what we call *completeness for strict language inclusion*.The second step is much more involved. There we exploit the Kleene theorem for Petri automata [11]: starting from expressions e, f such that ${}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f)$, we build two Petri automata $\mathscr{A}, \mathscr{B}$ respectively recognising $\mathcal{G}(e)$ and $\mathcal{G}(f)$. Then we design a product construction to synchronise $\mathscr{A}$ and $\mathscr{B}$, and a Kleene-like algorithm to extract from this construction two expressions e', f' such that $\mathcal{G}(e) = \mathcal{G}(e')$, $\mathsf{KL}^- \vdash e' \leq f'$, and $\mathcal{G}(f') \subseteq \mathcal{G}(f)$. This *synchronised Kleene theorem* suffices to conclude using the first step.

To our knowledge, this is the first completeness result for a theory involving Kleene iteration and intersection. Identity-free Kleene lattices were studied in depth by Andréka, Mikulás and Németi [2]; they have in particular shown that over this syntax, the equational theories generated by binary relations and formal languages coincide. But axiomatisability remained opened. The restriction to the identity-free fragment is important for several reasons. First of all, it makes it possible to rely on the technique used in [10] to compare Petri automata, which does not scale in the presence of identity. Second, this is the fragment for which the Kleene theorem for Petri automata is proved the most naturally [11]. Third, 'strange' laws appear in the presence of 1 [3], *e.g.*, $1 \cap (b \cdot a) \leq a \cdot (1 \cap (b \cdot a)) \cdot b$, and axiomatisability is still open even in the finitary case where Kleene iteration is absent – see the erratum about [3].

Proofs of completeness for other extensions of Kleene algebra include Kleene algebra with tests (KAT) [20], nominal Kleene algebra [24], and Concurrent Kleene algebra [27, 18]. The latter extension is the closest to our work since the parallel operator of concurrent Kleene algebra shares some properties of the intersection operation considered in the present work (*e.g.*, it is commutative and it satisfies a weak interchange law with sequential composition).

The paper is organised as follows. In Sect. 2, we recall KL^--expressions, their graph language and the corresponding model of Petri automata. In Sect. 3 we give our axiomatisation and state the completeness result. Then we show it following the proof scheme presented earlier: in Sect. 4 we show completeness for strict language inclusions, we recall in Sect. 5 the Kleene theorem of KL^- expressions, on which we build to show our synchronised Kleene theorem in Sect. 6.

2 Expressions, graph languages and Petri automata

2.1 Expressions and their relational semantics

We let $a, b \ldots$ range over the letters of a fixed alphabet X. We consider the following syntax of KL^--*expressions*, which we simply call expressions if there is no ambiguity:

$$e, f ::= e \cdot f \;\mid\; e + f \;\mid\; e \cap f \;\mid\; e^+ \;\mid\; 0 \;\mid\; a \qquad (a \in X)$$

We denote their set by Exp_X and we often write ef for $e \cdot f$. When we remove intersection ($\cap$) from the syntax of KL^--expressions we get KA^--*expressions*, which are the identity-free regular expressions.

If $\sigma : X \to \mathcal{P}(S \times S)$ is an interpretation of the letters into some space of relations, we write $\widehat{\sigma}$ for the unique homomorphism extending σ into a function from Exp_X to $\mathcal{P}(S \times S)$. An inequation between two expressions e and f is *valid*, written $\mathcal{R}el \models e \leq f$, if for every such interpretation σ we have $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$.

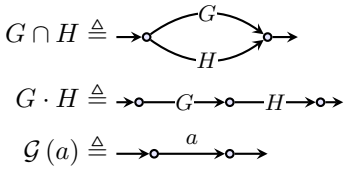


Figure 1 Operations on graphs.

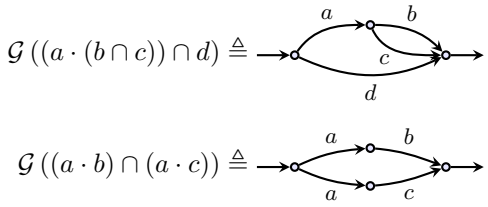


Figure 2 Graphs associated with some terms.

2.2 Terms, graphs, and homomorphisms

We let $u, v \ldots$ range over expressions built using only letters, $\cap$ and $\cdot$, which we call *terms*. (Terms thus form a subset of expressions: they are those expressions not using 0, $+$ and $\_^+$.) We will use 2-pointed labelled directed graphs, simply called *graphs* in the sequel. Those are tuples $\langle V, E, s, t, l, \iota, o\rangle$ with V (resp. E) a finite set of vertices (resp. edges), $s, t : E \to V$ the *source* and *target* functions, $l : E \to X$ the *labelling* function, and $\iota, o \in V$ two distinguished vertices, respectively called *input* and *output*.

As depicted in Fig. 1, graphs can be composed in series or in parallel, and a letter can be seen as a graph with a single edge labelled by that letter. One can thus recursively associate to every term u a graph $\mathcal{G}(u)$ called the *graph of* u. Two examples are given in Fig. 2; graphs of terms are *series-parallel* [31].

▶ **Definition 1** (Graph homomorphism). A *homomorphism* from $G = \langle V, E, s, t, l, \iota, o\rangle$ to $G' = \langle V', E', s', t', l', \iota', o'\rangle$ is a pair $h = \langle f, g\rangle$ of functions $f : V \to V'$ and $g : E \to E'$ that respect the various components: $s' \circ g = f \circ s$, $t' \circ g = f \circ t$, $l = l' \circ g$, $\iota' = f(\iota)$, and $o' = f(o)$.

We write $G' \lhd G$ if there exists a graph homomorphism from G to G'.

Such a homomorphism is depicted in Fig. 3. A pleasant way to think about graph homomorphisms is the following: we have $G \lhd H$ if G is obtained from H by merging (or identifying) some nodes, and by adding some extra nodes and edges. For instance, the graph G in Fig. 3 is obtained from H by merging the nodes $1, 2$ and by adding an edge between the input and the output labelled by d.

The starting point of the present work is the following characterisation:

▶ **Theorem 2** ([5, Thm. 1], [15, p. 208]). *For all terms* u, v, $\mathcal{R}el \models u \leq v$ *iff* $\mathcal{G}(u) \lhd \mathcal{G}(v)$.

2.3 Graph language of an expression

To generalise the previous characterisation to KL^--expressions, one interprets expressions by sets (languages) of graphs: graphs play the role of words for KA-expressions.

▶ **Definition 3** (Term and graph languages of expressions). The *term language* of an expression e, written $[\![e]\!]$, is the set of terms defined recursively as follows:

$$[\![e \cdot f]\!] \triangleq \{u \cdot v \mid u \in [\![e]\!] \text{ and } v \in [\![f]\!]\} \qquad [\![0]\!] \triangleq \emptyset$$
$$[\![e \cap f]\!] \triangleq \{u \cap v \mid u \in [\![e]\!] \text{ and } v \in [\![f]\!]\} \qquad [\![a]\!] \triangleq \{a\}$$
$$[\![e + f]\!] \triangleq [\![e]\!] \cup [\![f]\!] \qquad [\![e^+]\!] \triangleq \bigcup_{n>0} \{u_1 \cdot \cdots \cdot u_n \mid \forall i, u_i \in [\![e]\!]\}$$

The *graph language* of e is the set of graphs $\mathcal{G}(e) \triangleq \{\mathcal{G}(u) \mid u \in [\![e]\!]\}$.

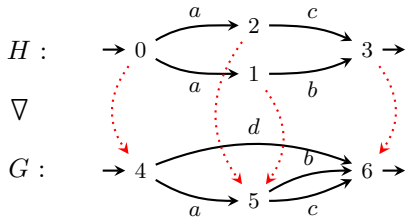


Figure 3 A graph homomorphism.

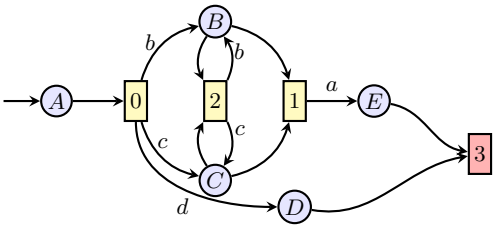


Figure 4 A Petri automaton.

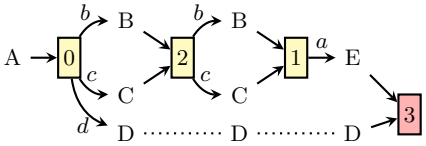


Figure 5 Run of a Petri automaton.

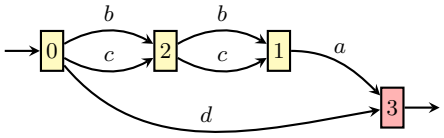


Figure 6 Graph of a run.

Note that for every term u, $[\![u]\!] = \{u\}$, so that the graph language of u thus contains just the graph of u. This justifies the overloaded notation $\mathcal{G}(u)$. Given a set S of graphs, we write ${}^{\lhd}S$ for its downward closure w.r.t. $\lhd$: ${}^{\lhd}S \triangleq \{G \mid G \lhd G', G' \in S\}$. We obtain:

▶ **Theorem 4** ([10, Thm. 6]). *For all expressions e, f, $\mathcal{R}el \models e \leq f$ iff ${}^{\lhd}\mathcal{G}(e) \subseteq {}^{\lhd}\mathcal{G}(f)$.*

2.4 Petri automata

We recall the notion of Petri automata [10, 11], an automata model that recognises precisely the graph languages of our expressions.

▶ **Definition 5** (Petri Automaton). A *Petri automaton (PA)* over the alphabet X is a tuple $\mathscr{A} = \langle P, \mathcal{T}, \iota \rangle$ where:

- P is a finite set of *places*,
- $\mathcal{T} \subseteq \mathcal{P}(P) \times \mathcal{P}(X \times P)$ is a set of *transitions*,
- $\iota \in P$ is the *initial place* of the automaton.

For each transition $t = \langle {}^{\triangleleft}t, t^{\triangleright} \rangle \in \mathcal{T}$, ${}^{\triangleleft}t$ is assumed to be non-empty; ${}^{\triangleleft}t \subseteq P$ is the *input of* t; and $t^{\triangleright} \subseteq X \times P$ is the *output of* t. We write $\pi_2(t^{\triangleright}) \triangleq \{p \mid \exists a, \langle a, p \rangle \in t^{\triangleright}\}$ for the set of the output places of t. Transitions with empty outputs are called *final*.

A PA is depicted in Fig. 4: places are represented by circles and transitions by squares.

Let us now recall the operational semantics of PA. Fix a PA $\mathscr{A} = \langle P, \mathcal{T}, \iota \rangle$ for the remainder of this section. A *state* of this automaton is a set of places. In a given state $S \subseteq P$, a transition $t = \langle {}^{\triangleleft}t, t^{\triangleright} \rangle$ is *enabled* if ${}^{\triangleleft}t \subseteq S$. In that case, we may fire t, leading to a new state $S' = (S \setminus {}^{\triangleleft}t) \cup \pi_2(t^{\triangleright})$. We write $S \xrightarrow{t}_{\mathscr{A}} S'$ in this case.

▶ **Definition 6** (Run of a PA). A *run* is a sequence $\langle S_1, t_1, S_2, \ldots, t_{n-1}, S_n \rangle$, where S_i are states, t_i are transitions such that $S_i \xrightarrow{t_i}_{\mathscr{A}} S_{i+1}$ for every $i \in [1, n-1]$, $S_1 = \{\iota\}$ and $S_n = \emptyset$.

A run of the PA from Fig. 4 is depicted in Fig. 5; this run gives rise to a graph, depicted in Fig. 6; see [11, Def. 3] for a formal definition in the general case.

▶ **Definition 7** (Graph language of a PA). The *graph language of a PA* $\mathscr{A}$, written $\mathcal{G}(\mathscr{A})$, consists of the graphs of its runs.

$$e \cap (f \cap g) = (e \cap f) \cap g \qquad e \cap f = f \cap e \qquad e \cap e = e$$

$$e \cap (f + g) = (e \cap f) + (e \cap g) \qquad e \cap (e + f) = e \qquad e + (e \cap f) = e$$

$$e + (f + g) = (e + f) + g \qquad e + f = f + e \qquad e + e = e$$

$$e{\cdot}(f{\cdot}g) = (e{\cdot}f){\cdot}g \quad e{\cdot}(f{+}g) = e{\cdot}f{+}e{\cdot}g \quad (e{+}f){\cdot}g = e{\cdot}g{+}f{\cdot}g \quad e{+}0 = e \quad e{\cdot}0 = 0 = 0{\cdot}e$$

$$e{+}e{\cdot}e^{+} = e^{+} = e{+}e^{+}{\cdot}e \quad e{\cdot}f{+}f = f \;\Rightarrow\; e^{+}{\cdot}f{+}f = f \quad f{\cdot}e{+}f = f \;\Rightarrow\; f{\cdot}e^{+}{+}f = f$$

Figure 7 KL^{-}: the first three lines correspond to distributive lattices, the last three to KA^{-}.

PA are assumed to be *safe* (in standard Petri net terminology, places contain at most one *token* at any time – whence the definition of states as sets rather than multisets) and to accept only series-parallel graphs. These two conditions are decidable [11]. Here we moreover assume that all PA have the same set of places P.

PA and KL^{-}-expressions denote the same class of graph languages:

▶ **Theorem 8** (Kleene theorem [11, Thm. 18])**.**

(i) *For every expression e, there is a Petri automaton $\mathscr{A}$ such that $\mathcal{G}(e) = \mathcal{G}(\mathscr{A})$.*

(ii) *Conversely, for every Petri automaton $\mathscr{A}$, there is an expression e such that $\mathcal{G}(e) = \mathcal{G}(\mathscr{A})$.*

3 Axiomatisation and structure of completeness proof

Let us introduce now our axiomatisation.

▶ **Definition 9.** The axioms of KL^{-} are the union of

- the axioms of identity-free Kleene algebra (KA^{-}) [23], and
- the axioms of a distributive lattice for $\{+, \cap\}$.

It is easy to check that those axioms are valid for binary relations, whence soundness of KL^{-}:

▶ **Theorem 10** (Soundness)**.** *If $\mathsf{KL}^{-} \vdash e \leq f$ then $\mathcal{R}el \models e \leq f$.*

The rest the paper is devoted the converse implication, which thanks to Thm. 4 amounts to:

▶ **Theorem 11** (Completeness)**.** *If ${}^{\triangleleft}\mathcal{G}(e) \subseteq {}^{\triangleleft}\mathcal{G}(f)$ then $\mathsf{KL}^{-} \vdash e \leq f$.*

The following very weak form of Thm. 11 is easy to obtain from the results in the literature:

▶ **Proposition 1.** *For all terms u, v, $\mathcal{G}(u) \lhd \mathcal{G}(v)$ entails $\mathsf{KL}^{-} \vdash u \leq v$.*

Proof. Follows from Thm. 4, completeness of semilattice-ordered semigroups [4] for relational models, and the fact the the axioms of KL^{-} entail those of semilattice-ordered semigroups. ◀

As explained in the introduction, our first step consists in proving KL^{-} completeness w.r.t. strict graph language inclusions, *i.e.*, not modulo homomorphisms:

▶ **Theorem 12** (Completeness for strict language inclusions)**.** *If $\mathcal{G}(e) \subseteq \mathcal{G}(f)$ then $\mathsf{KL}^{-} \vdash e \leq f$.*

The proof is given in Sect. 4. Our second step is to get the following theorem (Sect. 6):

▶ **Theorem 13** (Synchronised Kleene Theorem). *If $\mathscr{A}, \mathscr{B}$ are PA such that ${}^{\triangleleft}\mathcal{G}(\mathscr{A}) \subseteq {}^{\triangleleft}\mathcal{G}(\mathscr{B})$, then there are expressions e, f such that:*

$$\mathcal{G}(\mathscr{A}) = \mathcal{G}(e), \quad \mathsf{KL}^- \vdash e \leq f, \quad \text{and} \quad \mathcal{G}(f) \subseteq \mathcal{G}(\mathscr{B}).$$

The key observation for the proof is that the state-removal procedure used to transform a PA into a KL^- expression is highly non-deterministic. When considering two PA at a time, one can use this flexibility in order to synchronise the computation of the two expressions, so that they become easier to compare axiomatically. The concrete proof is quite technical and requires us to first recall many concepts from the proof [11] of Thm. 8(ii) (Sect. 5); it heavily relies on both Thm. 12 and Prop. 1.

Completeness of KL^- follows using Thm. 8(i) and Thm. 12 as explained in the introduction.

4 Completeness for strict language inclusion

Recall that the graph language of an expression e, $\mathcal{G}(e)$, is defined as the set of graphs of the term language of e, $[\![e]\!]$. We first prove that KL^- is complete for term language inclusions:

▶ **Proposition 2.** *If $[\![e]\!] \subseteq [\![f]\!]$ then $\mathsf{KL}^- \vdash e \leq f$.*

Proof. We follow a technique similar to the one recently used in [24]. We consider the maximal KA^--subexpressions, and we compute the atoms of the Boolean algebra of word languages generated by those expressions. By KA^- completeness [19, 23], we get KA^- (and thus KL^-) proofs that those are equal to appropriate sums of atoms. We distribute the surrounding intersections over those sums and replace the resulting intersections of atoms by fresh letters. This allows us to proceed recursively (on the intersection-depth of the terms), using substitutivity to recover a KL^- proof of the starting inequality. ◀

The difference between the term language and the graph language is that intersection is interpreted as an associative and commutative operation in the latter. We bury this difference by defining a 'saturation' function s on KL^--expressions such that for all e,

$$(\dagger) \quad \mathsf{KL}^- \vdash s(e) = e, \qquad \text{and} \qquad (\ddagger) \quad [\![s(e)]\!] = \{u \mid \mathcal{G}(u) \in \mathcal{G}(e)\} \quad .$$

Intuitively, this function uses distributivity and idempotency of sum to replace all intersections appearing in the expression by the sum of all their equivalent presentations modulo associativity and commutativity. For instance, $s(a \cap (b \cap c))$ is a sum of twelve terms (six choices for the ordering times two choices for the parenthesing). Technically, one should be careful to expand the expression first by maximally distributing sums, in order to make all potential n-ary intersections apparent. For instance, $((a \cap b) + d) \cap c$ expands to $((a \cap b) \cap c) + (d \cap c)$ so that its saturation is a sum of twelve plus two terms. For the same reason, all iterations should be unfolded once: we unfold and expand $(a \cap b)^+ \cap c$ into $((a \cap b) \cap c) + ((a \cap b) \cdot (a \cap b)^+ \cap c)$ before saturating it. We finally obtain Thm. 12 using (‡), Prop. 2, and (†):

$$\mathcal{G}(e) \subseteq \mathcal{G}(f) \quad \Rightarrow \quad [\![s(e)]\!] \subseteq [\![s(f)]\!] \quad \Rightarrow \quad \mathsf{KL}^- \vdash s(e) \leq s(f) \quad \Rightarrow \quad \mathsf{KL}^- \vdash e \leq f$$

5 Kleene theorem for Petri automata

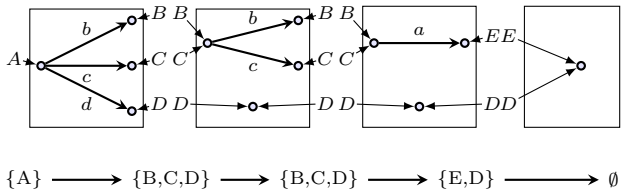
To prove the synchronised Kleene theorem (Thm. 13), we cannot use the Kleene theorem for PA (Thm. 8) as a black box: we use in a fine way the algorithm underlying the proof of the second item. We thus explain how it works [11] in details.

Figure 8 Rewriting rules for state-removal procedure.

Recall that to transform an NFA $\mathscr{A}$ to a regular expression e, one rewrites it using the rules of Fig. 8 until one reaches an automaton where there is a unique transition from the initial state to the final one, labelled by an expression e. While doing so, one goes through generalised NFA, whose transitions are labelled by regular expressions instead of letters.

We use the same technique for PA: we start by converting the PA into a NFA over a richer alphabet, which we call a *Template Automaton (TA)*, then we reduce this automaton using the rules of Fig. 8 until we get a single transition labelled by the desired expression.

To get some intuitions about the way we convert a PA into an NFA, consider the run in Fig. 5 and its graph in Fig. 6. One can decompose the run and the graph as follows:



The graph can thus be seen as a word over an alphabet of 'boxes', and the run as a path in an NFA whose states are sets of places of the PA. The letters of the alphabet, the above boxes, can be seen as 'slices of graphs'; they arise naturally from the transitions of the starting PA (Fig. 4 in this example).

5.1 Template automata

In order to make everything work, we need to refine both this notion of states and this notion of boxes to define template automata:

- states (sets of places) are refined into *types*. We let σ, τ range over types. A type is a tree whose leaves are labelled by places. When we forget the tree structure of a type τ, we get a a state $\overline{\tau}$. See [11, Def. 10] for a formal definition of types, which is not needed here. We call *singleton types* those types whose associated state is a singleton.
- letters will be *templates*: finite sets of boxes like depicted above but with edges labelled with arbitrary KL^--expressions; we define those formally below.

Given a directed acyclic graph (DAG) G, we write $\min G$ (resp. $\max G$) for the set of its sources (resp. sinks). A DAG is non-trivial when it contains at least one edge.

▶ **Definition 14** (Boxes). Let σ, τ be types. A *box* from σ to τ is a triple $\langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle$ where G is a non-trivial DAG with edges labelled in Exp_X, $\overrightarrow{\mathfrak{p}}$ is a map from $\overline{\sigma}$, the *input ports*, to the vertices of G, and $\overleftarrow{\mathfrak{p}}$ is a bijective map from $\overline{\tau}$, the *output ports*, to $\max G$, and where an additional condition relative to types holds [11, Def. 11]. (This condition can be kept abstract here.) A *basic* box is a box labelled with letters rather than arbitrary expressions. A *1-1* box is a box between singleton types.

We let α, β range over boxes and we write $\beta : \sigma \to \tau$ when β is a box from σ to τ.

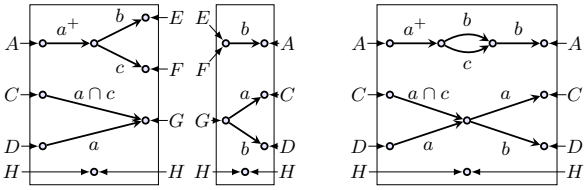


Figure 9 Two boxes and their composition.

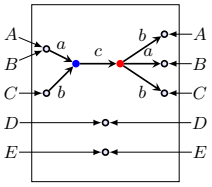


Figure 10 An atomic box.

We represent boxes graphically as in Fig. 9. Inside the rectangle is the DAG, with the input ports on the left-hand side and the output ports on the right-hand side. The maps $\overrightarrow{\mathfrak{p}}$ and $\overleftarrow{\mathfrak{p}}$ are represented by the arrows going from the ports to vertices inside the rectangle. Note that unlike $\overleftarrow{\mathfrak{p}}$, the map $\overrightarrow{\mathfrak{p}}$ may reach inner nodes of the DAG. 1-1 boxes are those with exactly one input port and one output port.

Boxes compose like in a category: if $\alpha : \sigma \to \tau$ and $\beta : \tau \to \rho$ then we get a box $\alpha \cdot \beta : \sigma \to \rho$ by putting the graph of α to the left of the graph of β, and for every port $p \in \overline{\tau}$, we identify the node $\overleftarrow{\mathfrak{p}_1}(p)$ with the node $\overrightarrow{\mathfrak{p}_2}(p)$. For instance the third box in Fig. 9 is obtained by composing the first two.

The key property enforced by the condition on types (kept abstract here) is the following:

▶ **Lemma 15.** *A 1-1 box is just a series-parallel 2-pointed graph labelled in* Exp_X.

Accordingly, one can extract a KL^--expression from any 1-1 box β, which we write $e(\beta)$ and call its *expression*.

▶ **Definition 16** (Templates). A *template* $\Gamma : \sigma \to \tau$ is a finite set of boxes from σ to τ. A *1-1 template* is a template of 1-1 boxes. The *expression* of a 1-1 template, written $e(\Gamma)$, is the sum of the expressions of its boxes.

Templates can be composed like boxes, by computing all pairwise box compositions.

▶ **Definition 17** (Box language of a template). A basic box is *generated* by a box β if it can be obtained by replacing each edge $x \xrightarrow{e} y$ of its DAG by a graph $G' \in \mathcal{G}(e)$ with input vertex x and output vertex y. The *box language* of a template Γ, written $\mathcal{B}(\Gamma)$, is the set of basic boxes generated by its boxes.

As expected, the box language of a template $\Gamma : \sigma \to \tau$ only contains boxes from σ to τ. Thanks to Lem. 15, when Γ is a 1-1 template, its box language can actually be seen as a set of graphs, and we have:

▶ **Proposition 3.** *For every 1-1 template* Γ*, we have* $\mathcal{B}(\Gamma) = \mathcal{G}(e(\Gamma))$.

We can finally define template automata:

▶ **Definition 18** (Template automaton (TA)). A *template automaton* is an NFA whose states are types, whose alphabet is the set of templates, whose transitions are of the form $\langle \sigma, \Gamma, \tau \rangle$ where $\Gamma : \sigma \to \tau$, and with a single initial state and a single accepting state which are singleton types. A *basic TA* is a TA whose all transitions are labelled by basic boxes.

By definition, a word accepted by a TA is a sequence of templates that can be composed into a single 1-1 template Γ, and thus gives rise to a set of graphs $\mathcal{B}(\Gamma)$. The *graph language of a TA* $\mathscr{E}$, written $\mathcal{G}(\mathscr{E})$, is the union of all those sets of graphs.

An important result of [11] is that we can translate every PA into a TA:

▶ **Proposition 4.** *For every PA $\mathscr{A}$, there exists a basic TA $\mathscr{E}$ such that $\mathcal{G}(\mathscr{A}) = \mathcal{G}(\mathscr{E})$.*

TA were defined so that they can be reduced using the state-removal procedure from Fig. 8. Templates can be composed sequentially and are closed under unions, so that now we only miss an operation $\_^*$ on templates to implement the first rule. Since we work in an identity-free (and thus star-free) setting, it suffices to define a strict iteration operation $\_^+$; and to rely on the following shorthands $\Delta \cdot \Gamma^* = \Delta \cup \Delta \cdot \Gamma^+$ and $\Gamma^* \cdot \Delta = \Delta \cup \Gamma^+ \cdot \Delta$.

Such an operation is provided in [11]:

▶ **Proposition 5.** *There exists a function $\_^+$ on templates such that if the TA obtained from a PA $\mathscr{A}$ through Prop. 4 reduces to a TA $\mathscr{E}$ by the rules in Fig. 8, then $\mathcal{G}(\mathscr{A}) = \mathcal{G}(\mathscr{E})$.* [1]

One finally obtains the Kleene theorem for PA by reducing the TA until it consists of a single transition labelled by a 1-1 template Γ: at this point, $e(\Gamma)$ is the desired KL^--expression.

5.2 Computing the iteration of a template

We need to know how the above template iteration can be defined to obtain our synchronised Kleene theorem, so that we explain it in this section. This section is required only to understand how we define a synchronised iteration operation in Sect. 6.

First notice that templates on which we need to compute $\_^+$ are of type $\sigma \to \sigma$. We first define this operation for a restricted class of templates, which we call *atomic*.

▶ **Definition 19** (Atomic boxes and templates, Support)**.** A box $\beta = \langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle : \sigma \to \sigma$ is *atomic* if its graph has a single non-trivial connected component C, and if for every vertex v outside C, there is a unique port $p \in \overline{\sigma}$ such that $\overrightarrow{\mathfrak{p}}(p) = \overleftarrow{\mathfrak{p}}(p) = v$. An *atomic template* is a template composed of atomic boxes.

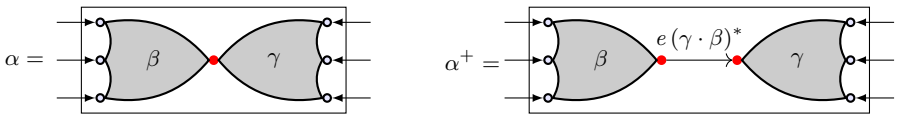
The *support* of a box $\beta : \sigma \to \sigma$ is the set $supp(\beta) \triangleq \{p \mid \overrightarrow{\mathfrak{p}}(p) \neq \overleftarrow{\mathfrak{p}}(p)\}$. The support of a template is the union of the supports of its boxes.

The following property of atomic boxes, makes it possible to compute their iteration:

▶ **Lemma 20** ([11, Lem. 7.18])**.** *The non-trivial connected component of an atomic box $\beta : \sigma \to \sigma$ always contains a vertex c, s.t. for every port p mapped inside that component, all paths from $\overrightarrow{\mathfrak{p}}(p)$ to a maximal vertex visit c. We call such a vertex a* bowtie *for β.*

Notice that the bowtie of a box is not unique. For instance, the atomic box in Fig. 10 contains two bowties: the blue and the red nodes.

We compute the iteration of an atomic box as follows. First choose a bowtie for this box, then split it at the level of this node into the product $\alpha = \beta \cdot \gamma$. The box $\gamma \cdot \beta$ is 1-1, we can thus extract from it a term $e(\gamma \cdot \beta)$. We set α^+ to be the template consisting of α and the box obtained from α by replacing the bowtie by an edge labelled $e(\gamma \cdot \beta)^+$. For the sake of conciseness, we denote this two-box template as on the right below, with an edge labelled with a starred expression.



[1] This statement is not simpler because, unfortunately, there is no function $\_^+$ on templates such that $\mathcal{B}(\Gamma^+) = \mathcal{B}(\Gamma)^+$).

Data: Atomic template Γ
Result: A template Γ^+ s.t.
$\mathcal{B}(\Gamma^+) = \mathcal{B}(\Gamma)^+$
if $\Gamma = \emptyset$ **then**
  Return $\emptyset$
else
  Write $\Gamma = \Delta \cup \{\alpha\} \cup \Sigma$ such that
  $supp(\Delta) \subseteq supp(\alpha)$ and
  $supp(\Sigma) \cap supp(\alpha) = \emptyset$;
  Choose a bowtie for α;
  Split α into $\beta \cdot \gamma$ at the level of this bowtie;
  Return
  $(\Delta^+ \cdot \Sigma^*) \cup (\Delta^* \cdot \Sigma^+) \cup (\Delta^* \cdot \delta \cdot \Delta^* \cdot \Sigma^*)$,
  where δ is the two-box template depicted on the right.
end

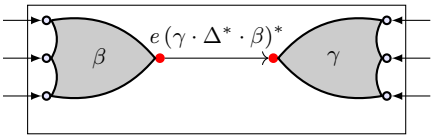


Figure 11 Iteration of an atomic template.

It is not difficult to see that $\mathcal{B}(\alpha^+) = \mathcal{B}(\alpha)^+$. Depending on the bowtie we have chosen, the box α^+ will be different. This is why we will write $\alpha^+_{\bowtie}$ to say that the bowtie $\bowtie$ has been selected for the computation of the iteration.

Now we need to generalise this construction to compute the iteration of an atomic template. For this, we need the following property, saying that the supports of atomic boxes of the same type are either disjoint or comparable:

▶ **Lemma 21.** *For all atomic boxes $\beta, \gamma : \sigma \to \sigma$, we have either 1) $supp(\beta) \subseteq supp(\gamma)$, or 2) $supp(\gamma) \subseteq supp(\beta)$, or 3) $supp(\beta) \cap supp(\gamma) = \emptyset$.*

We can compute the iteration of an atomic template by the algorithm in Fig. 11; intuitively, atomic boxes with disjoint support can be iterated in any order: they cannot interfere; in contrast, atomic boxes with small support must be computed before atomic boxes with strictly larger support: the iteration of the latter depends on that of the former. (Also note that since $supp(\Delta) \subseteq supp(\alpha)$ we have also $supp(\Delta^+) \subseteq supp(\alpha)$ thus the template $\gamma \cdot \Delta^* \cdot \beta$ is 1-1 and it gives rise to an expression $e(\gamma \cdot \Delta^* \cdot \beta)$.)

We finally compute the iteration of an arbitrary template $\Gamma : \sigma \to \sigma$ as follows: from each connected component of the graph of each box in Γ stems an atomic box; let $At(\Gamma)$ be the set of all these atomic boxes; we set $\Gamma^+ = At(\Gamma)^+$.

The overall algorithm contains two sources of non-determinism. First, one can partially choose in which order to process the atomic boxes. This is reflected by the choice of the box α, which we will call the *pivot*. For instance if $\Gamma = \{\alpha_1, \alpha_2, \beta\}$ such that $supp(\alpha_1) = supp(\alpha_2)$ and $supp(\beta) \cap supp(\alpha_1) = \emptyset$, then we can choose either α_1 or α_2 as the pivot, and the computation will respectively start with the computation of α_2^+ or that of α_1^+, yielding two distinct expressions. (In contrast, choices about boxes with disjoint support do not change the final result.) Second, every box of the template is eventually processed, and one must thus choose a bowtie for all of them. We write $\Gamma^+_{\bowtie,\leq}$ to make explicit the choice of the bowties and the computation order.

6 Synchronised Kleene theorem for PA

We can now prove Thm. 13. To synchronise the computation of two expressions e, f for two PA $\mathscr{A}, \mathscr{B}$ respectively, we construct a *synchronised product automaton* $\mathscr{E} \times \mathscr{F}$ between a TA $\mathscr{E}$ for $\mathscr{A}$ and a TA $\mathscr{F}$ for $\mathscr{B}$.

The states of this automaton are triples $\langle \sigma, \eta, \tau \rangle$ where σ and τ are types, *i.e.*, states from the TA $\mathscr{E}$ and $\mathscr{F}$, and $\eta : \overline{\tau} \to \overline{\sigma}$ is a function used to enforce coherence conditions. Its transitions have the form $\langle \langle \sigma, \eta, \tau \rangle, \langle \Gamma, \Delta \rangle, \langle \sigma', \eta', \tau' \rangle \rangle$ where $\langle \sigma, \Gamma, \sigma' \rangle$ is a transition of $\mathscr{E}$, $\langle \tau, \Delta, \tau' \rangle$ is a transition of $\mathscr{F}$, and Γ and Δ satisfy a certain condition which we call *refinement*, written $\Gamma \leq \Delta$.

The overall strategy is as follows. We reduce $\mathscr{E} \times \mathscr{F}$ using the rules of Fig. 8, where the operations $\cdot$ and $\cup$ are computed pairwise. The operation $\_^*$ is also computed pairwise, but in a careful way, exploiting the non-determinism of this operation to ensure that we maintain the refinement relation. We eventually get a single transition labelled by a pair of 1-1 templates Γ and Δ such that $\mathcal{B}(\Gamma) = \mathcal{G}(\mathscr{A})$, $\mathcal{B}(\Delta) = \mathcal{G}(\mathscr{B})$, and $\Gamma \leq \Delta$. To conclude, it suffices to deduce $\mathsf{KL}^- \vdash e(\Gamma) \leq e(\Delta)$ from the latter property. To sum-up, what we need to do now is:

- **Refinement**: define the refinement relation $\leq$ on templates;
- **Initialisation**: define $\mathscr{E} \times \mathscr{F}$ so that refinement holds;
- **Stability**: show that the refinement relation is maintained during the rewriting process;
- **Finalisation**: show that refinement between 1-1 templates entails KL^- provability.

6.1 Refinement relation

We first generalise graph homomorphisms to templates; this involves dealing with multiple ports, with finite sets, and with edge labels which are now arbitrary KL^--expressions. For the latter, we do not require strict equality but KL^--derivable inequalities.

▶ **Definition 22** (Box and template homomorphisms)**.** Let $\sigma, \tau, \sigma', \tau'$ be four types with two functions $\eta : \overline{\sigma} \to \overline{\tau}$ and $\eta' : \overline{\sigma'} \to \overline{\tau'}$. Let $\beta = \left\langle \overrightarrow{\mathfrak{p}}_\beta, \langle V_\beta, E_\beta, s_\beta, t_\beta, l_\beta \rangle, \overleftarrow{\mathfrak{p}}_\beta \right\rangle$ be a box of type $\tau \to \tau'$ and let $\alpha = \left\langle \overrightarrow{\mathfrak{p}}_\alpha, \langle V_\alpha, E_\alpha, s_\alpha, t_\alpha, l_\alpha \rangle, \overleftarrow{\mathfrak{p}}_\alpha \right\rangle$ be a box of type $\sigma \to \sigma'$. A homomorphism from α to β is a pair $\langle f, g \rangle$ of functions $f : V_\alpha \to V_\beta$ and $g : E_\alpha \to E_\beta$ s.t.:

- $s_\beta \circ g = f \circ s_\alpha$, $t_\beta \circ g = f \circ t_\alpha$,
- $\forall e \in E_\alpha, \quad \mathsf{KL}^- \vdash l_\beta \circ g(e) \leq l_\alpha(e)$,
- If $\{v\} \subseteq V_\alpha$ is a trivial connected component, so is $f(v)$.
- $\overrightarrow{\mathfrak{p}}_\beta \circ \eta = f \circ \overrightarrow{\mathfrak{p}}_\alpha$ and $\overleftarrow{\mathfrak{p}}_\beta \circ \eta' = f \circ \overleftarrow{\mathfrak{p}}_\alpha$. (We call this condition (η, η')-*compatibility*.)

We write $\beta \lhd_{\eta,\eta'} \alpha$ when there exists such a homomorphism. For two templates $\Gamma : \tau \to \tau'$ and $\Delta : \sigma \to \sigma'$, we write $\Gamma \lhd_{\eta,\eta'} \Delta$ if for all $\beta \in \Gamma$, there exists $\alpha \in \Delta$ such that $\beta \lhd_{\eta,\eta'} \alpha$.

We abbreviate $\Gamma \lhd_{\eta,\eta'} \Delta$ as $\Gamma \lhd \Delta$ when Γ, Δ are 1-1 templates, or when $\sigma = \tau$, $\sigma' = \tau'$ and η, η' are the identity function id. A box homomorphism is depicted in Fig. 12.

The above relation on templates is not enough for our needs; we have to extend it so that it is preserved during the rewriting process. We first write $\Gamma \sqsubseteq \Delta$ when $\mathcal{B}(\Gamma) \subseteq \mathcal{B}(\Delta)$, for two templates Γ, Δ of the same type. Refinement is defined as follows:

▶ **Definition 23** (Refinement)**.** We call *refinement* the relation on templates defined by $\leq_{\eta,\eta'} \triangleq \lhd_{\eta,\eta'} \cdot (\lhd_{\mathrm{id},\mathrm{id}} \cup \sqsubseteq)^*$, where $\_^*$ is reflexive transitive closure.

The following proposition shows that refinement implies provability of the expressions extracted from 1-1 templates. This gives us the finalisation step.

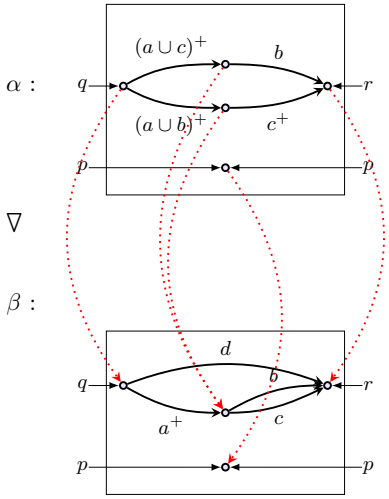


Figure 12 A box homomorphism.

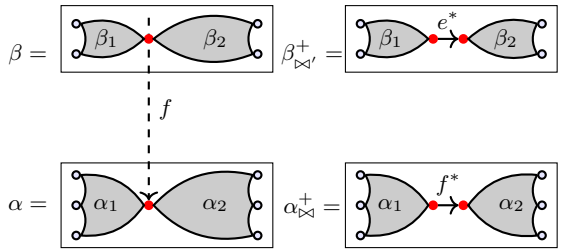


Figure 13 Bowtie compatible boxes.

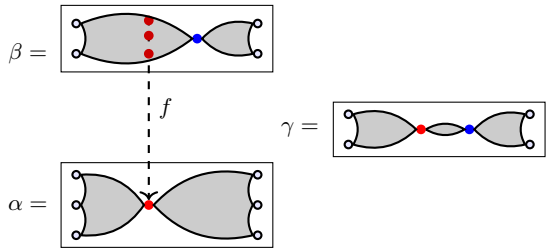


Figure 14 Case of bowtie incompatible boxes.

▶ **Proposition 6.** *If* Δ, Γ *are 1-1 templates such that* $\Delta \leq \Gamma$*, then* $\mathsf{KL}^- \vdash e(\Delta) \leq e(\Gamma)$.

Proof. When $\Delta \subseteq \Gamma$, it follows from Prop. 3 and Thm. 12; when $\Delta \lhd \Gamma$, it follows from Prop. 1. We conclude by transitivity. ◀

6.2 Synchronised product automaton (initialisation)

▶ **Definition 24** (2-Template automata (2-TA))**.** A *2-template automaton* is an NFA whose states are tuples of the form $\langle \tau, \eta, \sigma \rangle$ where τ, σ are types and $\eta : \overline{\sigma} \to \overline{\tau}$, whose alphabet is the set of pairs of templates, whose transitions are of the form $\langle \langle \sigma, \eta, \tau \rangle, \langle \Gamma, \Delta \rangle, \langle \sigma', \eta', \tau' \rangle \rangle$ where $\Gamma : \sigma \to \sigma'$, $\Delta : \tau \to \tau'$, and $\Gamma \leq_{\eta,\eta'} \Delta$, and with a single initial state and a single accepting state which consist of singleton types.

If $\mathcal{T}$ is a 2-TA, we denote by $\pi_1(\mathcal{T})$ (resp. $\pi_2(\mathcal{T})$) the automaton obtained by projecting the alphabet, the states and the transitions of $\mathcal{T}$ on the first (resp. last) component. Note that $\pi_1(\mathcal{T})$ and $\pi_2(\mathcal{T})$ are TA.

▶ **Definition 25** (Synchronised product of TA)**.** Let $\mathscr{E}, \mathscr{F}$ be two TA. The *synchronised product* of $\mathscr{E}$ and $\mathscr{F}$, written $\mathscr{E} \times \mathscr{F}$ is the 2-TA where $\langle \langle \tau, \eta, \sigma \rangle, \langle \Gamma, \Delta \rangle, \langle \tau', \eta', \sigma' \rangle \rangle$ is a transition of $\mathscr{E} \times \mathscr{F}$ iff $\langle \tau, \Gamma, \tau' \rangle$ is a transition of $\mathscr{E}$, $\langle \sigma, \Delta, \sigma' \rangle$ is a transition of $\mathscr{F}$ and $\Gamma \leq_{\eta,\eta'} \Delta$. (And with initial and accepting states defined from those of of $\mathscr{E}$ and $\mathscr{F}$.)

Note that we enforce refinement in the definition of this product, so that $\pi_1(\mathscr{E} \times \mathscr{F})$ is a sub-automaton of $\mathscr{E}$ and $\pi_2(\mathscr{E} \times \mathscr{F})$ is a sub-automaton of $\mathscr{F}$. Thus $\mathcal{G}(\pi_1(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{E})$ and $\mathcal{G}(\pi_2(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{F})$. When $\mathscr{E}, \mathscr{F}$ are TA coming from PA $\mathscr{A}, \mathscr{B}$ such that ${}^{\lhd}\mathcal{G}(\mathscr{A}) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{B})$, we can use the results from [11] about simulations to strengthen the first inclusion into an equality:

▶ **Theorem 26.** *Let* $\mathscr{A}, \mathscr{B}$ *be two PA,* $\mathscr{E}, \mathscr{F}$ *be basic TA such that* $\mathcal{G}(\mathscr{A}) = \mathcal{L}(\mathscr{E})$ *and* $\mathcal{G}(\mathscr{B}) = \mathcal{L}(\mathscr{F})$ *(given by Prop. 4). If* ${}^{\lhd}\mathcal{G}(\mathscr{A}_1) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{A}_2)$ *then:*

- $\mathcal{G}(\pi_1(\mathscr{E} \times \mathscr{F})) = \mathcal{G}(\mathscr{A})$;

- $\mathcal{G}(\pi_2(\mathscr{E} \times \mathscr{F})) \subseteq \mathcal{G}(\mathscr{B})$.

Proof. The second point follows from the observation above. The first one comes from the simulation result ([11, Prop. 9.10]) for PA. Indeed, if ${}^{\lhd}\mathcal{G}(\mathscr{A}) \subseteq {}^{\lhd}\mathcal{G}(\mathscr{B})$, then there is a simulation ([11, Def. 9.2]) between $\mathscr{A}$ and $\mathscr{B}$. This implies that for every run $\langle \tau_1, \Gamma_1, \tau_2, \dots, \Gamma_{n-1}, \tau_n \rangle$ of $\mathscr{E}$, there is a run $\langle \sigma_1, \Delta_1, \sigma_2, \dots, \Delta_{n-1}, \sigma_n \rangle$ of $\mathscr{F}$ and a set of mapping $\eta_i : \overline{\sigma_i} \to \overline{\tau_i}$, $i \in [1, n]$ such that $\Gamma_i \lhd_{\eta_i, \eta_{i+1}} \Delta_i$ for every $i \in [1, n-1]$. ◀

6.3 Maintaining refinement during reductions

Let us finally show that refinement is stable by composition, union, and iteration.

▶ **Theorem 27** (Stability of refinement by $\cdot$ and $\cup$)**.**
- *If* $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ *and* $\Delta_1 \leq_{\eta',\eta"} \Delta_2$ *then* $\Gamma_1 \cdot \Delta_1 \leq_{\eta,\eta"} \Gamma_2 \cdot \Delta_2$.
- *If* $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ *and* $\Delta_1 \leq_{\eta,\eta'} \Delta_2$ *then* $\Gamma_1 \cup \Delta_1 \leq_{\eta,\eta'} \Gamma_2 \cup \Delta_2$.

Proof. To show the first property it suffices to show the following results:

$$\text{If} \quad \Gamma_1 \lhd_{\eta,\eta'} \Gamma_2 \quad \text{and} \quad \Delta_1 \lhd_{\eta',\eta''} \Gamma_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \lhd_{\eta',\eta''} \Gamma_2 \cdot \Delta_2. \qquad (L_1)$$

$$\text{If} \quad \Gamma_1 \sqsubseteq \Gamma_2 \quad \text{and} \quad \Delta_1 \sqsubseteq \Delta_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \sqsubseteq \Gamma_2 \cdot \Delta_2. \qquad (L_2)$$

$$\text{If} \quad \Gamma_1 \lhd \Gamma_2 \quad \text{and} \quad \Delta_1 \sqsubseteq \Delta_2 \quad \text{then} \quad \Gamma_1 \cdot \Delta_1 \, (\lhd \cdot \sqsubseteq)^* \, \Gamma_2 \cup \Delta_2. \qquad (L_3)$$

To show (L_1), consider a box $\alpha_1 \in \Gamma_1$ and $\beta_1 \in \Delta_1$. By hypothesis, there is a box $\alpha_2 \in \Gamma_2$ and an (η, η')-compatible homomorphism $h = \langle f, g \rangle$ from α_2 to α_1 and a box $\beta_2 \in \Delta_2$ and an (η', η'')-compatible homomorphism $h' = \langle f', g' \rangle$ from β_2 to β_1. Let $h'' = \langle f'', g'' \rangle$, where f'' equals f in $\mathrm{dom}(f)$ and f' in $\mathrm{dom}(f')$, and g'' equals g in $\mathrm{dom}(g)$ and g' in $\mathrm{dom}(g')$. Using (η, η')-compatibility of h and (η', η'')-compatibility of h', it is easy to show that h'' is an (η, η'')-compatible homomorphism from $\alpha_2 \cdot \beta_2$ to $\alpha_1 \cdot \beta_1$, which concludes the proof of (L_1). (L_2) follows easily from the definition of $\sqsubseteq$. For (L_3), note that $\Delta_1 \lhd \Delta_1$ (we choose the identity homomorphism), thus by (L_1), we have that $\Gamma_1 \cdot \Delta_1 \lhd \Gamma_2 \cdot \Delta_1$. By (L_2), we have that $\Gamma_2 \cdot \Delta_1 \sqsubseteq \Gamma_2 \cdot \Delta_2$, which concludes the proof.

To show the first property, we proceed by induction on the length of the sequences justifying that $\Gamma_1 \leq_{\eta,\eta'} \Gamma_2$ and $\Delta_1 \leq_{\eta',\eta"} \Delta_2$, using $(L_1), (L_2)$ and (L_3) for the base cases.

To show the second property, we follow the same proof schema, showing results similar to $(L_1) - (L_3)$ where $\cdot$ is replaced by $\cup$. ◀

▶ Remark. Thm. 27 justifies our definition of $\leq_{\eta,\eta'}$. Indeed, a more permissive definition would seem natural, but the first property of Thm 27 would fail. For instance, if $\Gamma_1 \sqsubseteq \Gamma_2$ and $\Delta_1 \lhd_{\eta,\eta'} \Delta_2$, we do not have in general that $\Gamma_1 \cdot \Delta_1 \leq_{\eta,\eta'} \Gamma_2 \cdot \Delta_2$.

The main theorem of this section is Thm 28, stating that the refinement relation is stable under iteration. As its proof is very technical, we give only a proof sketch here, and leave the technical details to [13, App. B].

▶ **Theorem 28** (Stability of refinement by $\_^+$)**.** *If* $\Gamma \leq_{\eta,\eta} \Delta$ *then there are bowtie choices* $\bowtie, \bowtie'$ *and computation orders* $\preceq, \preceq'$*, for* Γ *and* Δ *respectively, such that:* $\Gamma^+_{\bowtie,\preceq} \leq_{\eta,\eta} \Delta^+_{\bowtie',\preceq'}$.

Proof sketch. To prove Thm. 28, it is enough to show the following properties.
- If $\Gamma \sqsubseteq \Delta$ then, for every bowtie choices $\bowtie, \bowtie'$, and every computation orders $\preceq, \preceq'$ for Γ and Δ respectively, we have that $\Gamma^+_{\bowtie,\preceq} \sqsubseteq \Delta^+_{\bowtie',\preceq'}$.
- If $\Gamma \lhd_{\eta,\eta} \Delta$ then there are two bowtie choices $\bowtie, \bowtie'$ and two computation orders $\preceq, \preceq'$, for Γ and Δ respectively, such that $\Gamma^+_{\bowtie,\preceq} \leq_{\eta,\eta} \Delta^+_{\bowtie',\preceq'}$.

The first property follows from $\mathcal{B}(\Gamma^+_{\bowtie,\preceq}) = \mathcal{B}(\Gamma)^+$ for every bowtie choice $\bowtie$ and order $\preceq$.

For the sake of clarity, we give here the proof of the second proposition in the case where Γ and Δ are singletons of atomic boxes $\{\alpha\}$ and $\{\beta\}$ respectively. The general case is treated in [13, App. B]. Let $\bowtie, \bowtie'$ be bowtie choices for α and β respectively, and let $h = \langle f, g\rangle$ be a homomorphism from β to α.

Let us first treat the case where $f^{-1}(\bowtie) = \{\bowtie'\}$ (we say that α, β are bowtie compatible). This is illustrated by the boxes α, β of Fig. 13, where the bowties are the red nodes. If we decompose α and β at the level of their bowties, we get $\alpha = \alpha_1 \cdot \alpha_2$ and $\beta = \beta_1 \cdot \beta_2$, where $\alpha_2 \cdot \alpha_1$ and $\beta_2 \cdot \beta_1$ are 1-1 boxes. We write $e = e\,(\alpha_2 \cdot \alpha_1)$ and $f = e\,(\beta_2 \cdot \beta_1)$. The boxes $\alpha^+_{\bowtie}$ and $\beta^+_{\bowtie'}$ are depicted in Fig. 13. Let us show that there is a homomorphism from $\beta^+_{\bowtie'}$ to $\alpha^+_{\bowtie}$. The homomorphism h induces a homomorphism h_1 from β_1 to α_1 and a homomorphism h_2 from β_2 to α_2 ([13, App. B, Lem. 42]). Combining h_1 and h_1, we get almost a homomorphism from $\beta^+_{\bowtie'}$ to $\alpha^+_{\bowtie}$ (See Fig. 13), we need only to show that $\mathsf{KL}^- \vdash e \leq f$. But this follows from Prop. 6: indeed, we can combine h_1 and h_2 to get a homomorphism from $\beta_2 \cdot \beta_1$ to $\alpha_2 \cdot \alpha_1$. We have thus that $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \beta^+_{\bowtie'}$ ((η,η)-compatibility is easy).

Let us now treat the case where $N := f^{-1}(\bowtie)$ is not necessarily $\{\bowtie'\}$ (N is illustrated by the red node of β in Fig. 14). Let γ be the box obtained from β by merging the nodes N (see Fig. 14). There are two bowtie choices for γ: a bowtie $\bowtie_b$ inherited from β (blue in Fig. 14) and a bowtie $\bowtie_r$ coming from the nodes of N (red in Fig. 14).

Let h' be the homomorphism from β to γ that maps each node (and each edge) to itself, except for the nodes of N which are mapped to $\bowtie_r$. If we consider the bowtie $\bowtie_b$ for γ, then β and γ are bowtie compatible w.r.t. to h', thus $\gamma^+_{\bowtie_b} \lhd \beta^+_{\bowtie'}$ using the previous case.

Let h'' be the homomorphism from γ to α, which is exactly h except that it maps the node $\bowtie_r$ to the bowtie $\bowtie$ of α. If we consider the bowtie $\bowtie_r$ for γ, then γ and α are bowtie compatible w.r.t. h'', thus $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \gamma^+_{\bowtie_r}$ using the previous case again.

Notice finally that $\gamma^+_{\bowtie_r} \sqsubseteq \gamma^+_{\bowtie_b}$. To sum up, we have: $\alpha^+_{\bowtie} \lhd_{\eta,\eta} \gamma^+_{\bowtie_r} \sqsubseteq \gamma^+_{\bowtie_b} \lhd \beta^+_{\bowtie'}$. ◀

The last case in this proof explains the need to work with refinement ($\leq$) rather than just homomorphisms ($\lhd$): when starting from templates that are related by homomorphism and iterating them, the templates we obtain are not necessarily related by a single homomorphism, only by a sequence of homomorphisms and inclusions.

7 Future work

We have proven that KL^- axioms are sound and complete w.r.t. the relational models of identity-free Kleene lattices, and thus also w.r.t. their language theoretic models, by the results from [2].

Whether one can obtain a finite axiomatisation in presence of identity remains open. This question is important since handling the identity relation is the very first step towards handling *tests*, which are crucial in order to model the control flow of sequential programs precisely (*e.g.*, as in Kleene algebra with tests [20]).

An intermediate problem, which is still open to the best of our knowledge, consists in finding an axiomatisation for the fragment with composition, intersection and identity (not including transitive closure) [3, see errata available online].

References

1 C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: semantic foundations for networks. In *POPL*, pages 113–126. ACM, 2014. doi:10.1145/2535838.2535862.

2 H. Andréka, S. Mikulás, and I. Németi. The equational theory of Kleene lattices. *Theoretical Computer Science*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.

3 H. Andréka and Sz. Mikulás. Axiomatizability of positive algebras of binary relations. *Algebra Universalis*, 66(1):7–34, 2011. doi:10.1007/s00012-011-0142-3.

4 H. Andréka. Representation of distributive lattice-ordered semigroups with binary relations. *Algebra Universalis*, 28:12–25, 1991.

5 H. Andréka and D.A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.

6 A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, CS Dpt., Cornell University, July 2001. URL: http://hdl.handle.net/1813/5831.

7 A. Armstrong, G. Struth, and T. Weber. Programming and automating mathematics in the Tarski-Kleene hierarchy. *Journal of Logical and Algebraic Methods in Programming*, 83(2):87–102, 2014. doi:10.1016/j.jlap.2014.02.001.

8 Maurice Boffa. Une condition impliquant toutes les identités rationnelles. *Informatique Théorique et Applications*, 29(6):515–518, 1995. URL: http://archive.numdam.org/.../ITA_1995__29_6_515_0.pdf.

9 Thomas Braibant and Damien Pous. Deciding Kleene algebras in Coq. *Logical Methods in Computer Science*, 8(1):1–16, 2012. doi:10.2168/LMCS-8(1:16)2012.

10 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS*, pages 68–79. ACM, 2015. doi:10.1109/LICS.2015.17.

11 Paul Brunet and Damien Pous. Petri automata. *Logical Methods in Computer Science*, Volume 13, Issue 3, 2017. doi:10.23638/LMCS-13(3:33)2017.

12 J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.

13 Amina Doumane and Damien Pous. Completeness for identity-free kleene lattices. Full version of this extended abstract, available at https://hal.archives-ouvertes.fr/hal-01780845, 2018. URL: https://hal.archives-ouvertes.fr/hal-01780845.

14 S. Foster, G. Struth, and T. Weber. Automated engineering of relational and algebraic methods in Isabelle/HOL - (invited tutorial). In *RAMiCS*, volume 6663 of *LNCS*, pages 52–67. Springer, 2011. doi:10.1007/978-3-642-21070-9_5.

15 P.J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland. Elsevier, 1990.

16 C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In *CONCUR*, volume 5710 of *LNCS*, pages 399–414. Springer, 2009. doi:10.1007/978-3-642-04081-8_27.

17 P. Höfner and G. Struth. On automating the calculus of relations. In *IJCAR*, volume 5195 of *LNCS*, pages 50–66. Springer, 2008. doi:10.1007/978-3-540-71070-7_5.

18 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent kleene algebra: Free model and completeness. In *ESOP*, volume 10801 of *LNCS*, pages 856–882. Springer, 2018. doi:10.1007/978-3-319-89884-1_30.

19 D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

20 D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.

21 D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

22 D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *CL2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582. Springer, 2000. doi:10.1007/3-540-44957-4_38.

23 Dexter Kozen. Typed Kleene algebra. Technical Report TR98-1669, CS Dpt., Cornell University, 1998. URL: http://www.cs.cornell.edu/~kozen/papers/typed.pdf.

24 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in nominal kleene algebra. *J. Log. Algebr. Meth. Program.*, 91:17–32, 2017. doi:10.1016/j.jlamp.2017.06.002.

25 A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Algebraic Reasoning*, 49(1):95–106, 2012. doi:10.1007/s10817-011-9223-4.

26 D. Krob. Complete systems of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.

27 Michael R. Laurence and Georg Struth. Completeness theorems for pomset languages and concurrent kleene algebras. *CoRR*, abs/1705.05896, 2017. arXiv:1705.05896.

28 Damien Pous. Kleene Algebra with Tests and Coq tools for while programs. In *ITP*, volume 7998 of *LNCS*, pages 180–196. Springer, 2013. doi:10.1007/978-3-642-39634-2_15.

29 V. R. Pratt. Dynamic algebras and the nature of induction. In *STOC*, pages 22–28. ACM, 1980. doi:10.1145/800141.804649.

30 Volodimir Nikiforovych Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964.

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