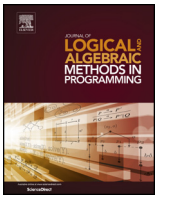
Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Algorithms for Kleene algebra with converse ☆

Paul Brunet, Damien Pous *,[1]

Plume team – LIP, CNRS, ENS de Lyon, Université de Lyon, UMR 5668, France

ARTICLE INFO

Article history:
Received 1 October 2014
Received in revised form 5 May 2015
Accepted 17 July 2015
Available online xxxx

Keywords:
Kleene algebra
Converse
Automata
Algorithm
Regular expressions

ABSTRACT

The equational theory generated by all algebras of binary relations with operations of union, composition, converse and reflexive transitive closure was studied by Bernátsky, Bloom, Ésik and Stefanescu in 1995. In particular, they obtained its decidability by using a particular automata construction.

We show that deciding this equational theory is PSPACE-complete, by providing a PSPACE algorithm (the problem is easily shown to be PSPACE-hard). We obtain other algorithms that are time-efficient in practice, despite not being PSPACE.

Our results use an alternative automata construction, inspired by the one from Bloom, Ésik and Stefanescu. We relate those two constructions by exhibiting a bisimulation between the resulting deterministic automata, and by showing how our construction results in more sharing between states, thus producing smaller automata.

1. Introduction

In many contexts in computer science and mathematics operations of union, sequence or product and iteration appear naturally. *Kleene Algebra*, introduced by John H. Conway under the name *regular algebra* [2], provide an algebraic framework allowing to express properties of these operators, by studying the equivalence of expressions built with these connectives. It is well known that the corresponding equational theory is decidable [3], and that it is complete for both language and relation models.

As expressive as it may be, one may wish to integrate other usual operations in such a setting. Theories obtained this way, by addition of a finite set of equations to the axioms of Kleene Algebra, are called *Extensions of Kleene Algebra*. We shall focus here on one of these extensions, where an operation of *converse* is added to Kleene Algebra. The converse of a word is its mirror image (the word obtained by reversing the order of the letters), and the converse $R^\vee$ of a relation R is its reciprocal ($x\, R^\vee\, y \triangleq y\, R\, x$). This natural operation can be expressed simply as a set of equations that we add to Kleene Algebra's axioms.

The question that arises once this theory is defined is its decidability: given two formal expressions built with the connectives product, sum, iteration and converse, can one decide automatically if they are equivalent, meaning that their equality can be proven using the axioms of the theory? Bloom, Ésik, Stefanescu and Bernátsky gave an affirmative answer to that question in two articles, [4] and [5], in 1995.

☆ Extended version of the abstract presented at RAMiCS '2014 [1].

* Corresponding author.
E-mail address: damien.pous@ens-lyon.fr (D. Pous).

[1] Work partially funded by the french projects PiCoq (ANR-09-BLAN-0169-01) and PACE (ANR-12IS02001).

http://dx.doi.org/10.1016/j.jlamp.2015.07.005
2352-2208/

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

However, although the algorithm they define proves the decidability result, it is too costly (in terms of time and memory consumption) to be used in concrete applications. In this paper, beside some simplifications of the proofs given in [4], we give a new and more efficient algorithm to decide this problem, which we place in the complexity class PSPACE.

The equational theory of Kleene algebra cannot be finitely axiomatised [6]. Krob presented the first purely axiomatic (but infinite) presentation [7]. Several finite quasi-equational characterisations have been proposed [8,9,7,10,11]; here we follow the one from Kozen [10].

A Kleene Algebra is an algebraic structure $\langle K, +, \cdot, ^\star, \mathbb{0}, \mathbb{1}\rangle$ such that $\langle K, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ is an idempotent semiring, and the operation * satisfies the following axioms. (Here $a \leqslant b$ is a shorthand for $a + b = b$.)

$$\mathbb{1} + a \cdot a^\star \leqslant a^\star \tag{1a}$$

$$\mathbb{1} + a^\star \cdot a \leqslant a^\star \tag{1b}$$

$$b + a \cdot x \leqslant x \Rightarrow a^\star \cdot b \leqslant x \tag{1c}$$

$$b + x \cdot a \leqslant x \Rightarrow b \cdot a^\star \leqslant x \tag{1d}$$

The quasi-variety KA consists of the axioms of an idempotent semiring together with axioms and implications (1a) to (1d). Kleene Algebras are thus *models* of KA. We shall call *regular expressions over X*, written Reg_X, the expressions built from letters of X, the binary connectives $+$ and $\cdot$, the unary connective * and the two constants $\mathbb{0}$ and $\mathbb{1}$.

Two families of such algebras are of particular interest: languages (sets of finite words over a finite alphabet, with union as sum and concatenation as product) and relations (binary relations over an arbitrary set with union and composition). KA is complete for both these models [7,10], meaning that for any $e, f \in \mathrm{Reg}_X$, $\mathrm{KA} \vdash e = f$ if and only if e and f coincide under any language (resp. relational) interpretation. This last property will be written $e \equiv_{\mathrm{Lang}} f$ (resp. $e \equiv_{\mathrm{Rel}} f$).

More remarkably, if we denote by $[\![e]\!]$ the language denoted by an expression e, we have that for any $e, f \in \mathrm{Reg}_X$, $\mathrm{KA} \vdash e = f$ if and only if $[\![e]\!] = [\![f]\!]$. By Kleene's theorem (see [3]) the equality of two regular languages can be reduced to the equivalence of two finite automata, which is decidable. Hence, the theory KA is decidable.

Now let us add a unary operation of converse to regular expressions. We shall denote by $\mathrm{Reg}_X^\vee$ the set of regular expressions with converse over a finite alphabet X. While doing so, several questions arise:

1. Can the converse on languages and on relations be encoded in the same theory?
2. What axioms do we need to add to KA to model these operations?
3. Are the resulting theories complete for languages and relations?
4. Are these theories decidable?

There is a simple answer to the first question: no. Indeed the equation $a \leqslant a \cdot a^\vee \cdot a$ is valid for any relation a (because if $(x, y) \in a$, then $(x, y) \in a$, $(y, x) \in a^\vee$, and $(x, y) \in a$, so that $(x, y) \in a \cdot a^\vee \cdot a$). But this equation is not satisfied for all languages a (for instance, with the language $a = \{x\}$, $a \cdot a^\vee \cdot a = \{xxx\}$ and $x \notin \{xxx\}$). This means that there are two distinct theories corresponding to these two families of models. Let us begin by considering the case of languages.

Theorem 1 *(Completeness of KAC$^-$). (See [4].) A complete axiomatisation of the variety* $\mathrm{Lang}^\vee$ *of languages generated by concatenation, union, star, and converse consists of the axioms of KA together with axioms* (2a) *to* (2d).

$$(a + b)^\vee = a^\vee + b^\vee \tag{2a}$$

$$(a \cdot b)^\vee = b^\vee \cdot a^\vee \tag{2b}$$

$$(a^\star)^\vee = (a^\vee)^\star \tag{2c}$$

$$a^{\vee\vee} = a \tag{2d}$$

We call this theory KAC$^-$; it is decidable.

As before, we write $e \equiv_{\mathrm{Lang}^\vee} f$ if e and f have the same language interpretations (for a formal definition, see the "Notation" subsection below). To prove this result, one first associates to any expression $e \in \mathrm{Reg}_X^\vee$ an expression $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$, where $\mathbf{X}$ is an alphabet obtained by adding to X a disjoint copy of itself. Then, one proves that the following implications hold.

$$e \equiv_{\mathrm{Lang}^\vee} f \quad \Rightarrow \quad [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \tag{3}$$

$$[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \quad \Rightarrow \quad \mathrm{KAC}^- \vdash e = f \tag{4}$$

(That $\mathrm{KAC}^- \vdash e = f$ entails $e \equiv_{\mathrm{Lang}^\vee} f$ is obvious; decidability comes from that of regular languages equivalence.) We reformulate Bloom et al.'s proofs of these implications in elementary terms in Section 2.1.

As stated before, the equation $a \leqslant a \cdot a^\vee \cdot a$ provides a difference between languages with converse and relations with converse. It turns out that it is the only difference, in the sense that the following theorem holds:

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

Theorem 2 *(Completeness of KAC). (See [4,5].) A complete axiomatisation of the variety* $\mathrm{Rel}^\vee$ *of relations generated by composition, union, star, and converse consists of the axioms of* KAC^- *together with the axiom* (5).

$$a \leqslant a \cdot a^\vee \cdot a \tag{5}$$

We call this theory KAC; it is decidable.

The proof of this result also relies on a translation into regular languages. Ésik et al. define a notion of *closure*, written $cl()$, for languages over $\mathbf{X}$, and they prove the following implications:

$$e \equiv_{\mathrm{Rel}^\vee} f \quad \Rightarrow \quad cl([\![\mathbf{e}]\!]) = cl([\![\mathbf{f}]\!]) \tag{6}$$

$$cl([\![\mathbf{e}]\!]) = cl([\![\mathbf{f}]\!]) \quad \Rightarrow \quad \mathrm{KAC} \vdash e = f \tag{7}$$

(Again, that $\mathrm{KAC} \vdash e = f$ entails $e \equiv_{\mathrm{Rel}^\vee} f$ is obvious.) The first implication (6) was proven in [4]; we give a new formulation of this proof in Section 2.2. The second implication (7) was proven in [5]. We show in Section 2.3 how to relate the above notion of closure with a more general notion used independently by Freyd and Scedrov [12] and Andréka and Bredikhin [13].

The last consideration is the decidability of KAC. To this end, Bloom et al. propose a construction to obtain an automaton recognising $cl(L)$, when given an automaton recognising L. Decidability follows: to decide whether $\mathrm{KAC} \vdash e = f$ one can build two automata recognising $cl([\![\mathbf{e}]\!])$ and $cl([\![\mathbf{f}]\!])$ and check if they are equivalent. Unfortunately, their construction tends to produce huge automata, which makes it inappropriate for practical applications. We propose a new and simpler construction in Section 4, which we analyse in Section 5:

- we compare it to Bloom et al.'s construction by exhibiting a bisimulation relation and by showing how our construction makes it possible to share more states (Section 5.1);
- we give a simple bound on the size of the produced automata (Section 5.2);
- we use this bound to provide a PSPACE algorithm, to deduce that the problem of equivalence in KAC is PSPACE-complete (Section 5.3);
- we finally provide algorithms that are not PSPACE but time-efficient in practice, using "up to techniques" [14,15] for bisimulations (Section 5.4).

1.1. Notation

For any word w, $|w|$ is the size of w, meaning its number of letters; for any $1 \leqslant i \leqslant |w|$, we write $w(i)$ for the ith letter of w and $w|_i \triangleq w(1)w(2)\cdots w(i)$ for its prefix of size i. Also, $\mathrm{suffixes}(w) \triangleq \{v \mid \exists u : uv = w\}$ is the set of all suffixes of w. A deterministic automaton is a tuple $\langle Q, \Sigma, q_0, \mathscr{T}, \delta\rangle$; with Q a set of states, Σ an alphabet, $q_0 \in Q$ an initial state, $\mathscr{T} \subseteq Q$ a set of final states and $\delta : Q \times \Sigma \to Q$ a transition function. A non-deterministic automaton is a tuple $\langle Q, \Sigma, I, \mathscr{T}, \Delta\rangle$; with Q, Σ and $\mathscr{T}$ same as before, $I \subseteq Q$ a set of initial states and $\Delta \subseteq Q \times \Sigma \times Q$ a set of transitions. We write $L(\mathscr{A})$ for the *language recognised by the automaton* $\mathscr{A}$. For any $a \in \Sigma$, we write $\Delta(a)$ for the relation induced on Q by a, defined as: $\{(p,q) \mid (p,a,q) \in \Delta\}$. We also use the compact notation $p \xrightarrow{w}_{\mathscr{A}} q$ to denote that there is in the automaton $\mathscr{A}$ a path labelled by w from the state p to the state q. We denote by $R \cdot S$ the composition of two relations: $R \cdot S \triangleq \{(x,y) \mid \exists z, x\,R\,z \wedge z\,R\,y\}$. For a set $E \subseteq Q$ and a relation R over Q, we write $E \cdot R$ for the set $\{y \mid \exists x \in E, x\,R\,y\}$.

Given a map σ from a set X to the languages on an alphabet Σ (resp. the relations on a set S), there is a unique extension of σ into a homomorphism from Reg_X to Lang_Σ (resp. Rel_S), which we denote by $\widehat{\sigma}$. The same extensions exist for regular expressions with converse, and we will use the same notation for them. We finally denote by $\equiv_V$ the equality in a variety V (Lang, Rel, $\mathrm{Lang}^\vee$ or $\mathrm{Rel}^\vee$): $e \equiv_V f \triangleq \forall K, \forall \sigma : X \to V_K, \widehat{\sigma}(e) = \widehat{\sigma}(f)$.

2. Preliminary material

2.1. Languages with converse: theory KAC^-

We consider regular expressions with converse over a finite alphabet X. The alphabet $\mathbf{X}$ is defined as $X \cup X'$, where $X' \triangleq \{x' \mid x \in X\}$ is a disjoint copy of X. As a shorthand, we use $'$ as an internal operation on $\mathbf{X}$ going from X to X' and from X' to X such that if $x \in X$, $x' \triangleq x' \in X'$ and $(x')' \triangleq x \in X$. An important operation in the following is the translation of an expression $e \in \mathrm{Reg}_X^\vee$ to an expression $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$. We proceed to its definition in two steps.

Let $\tau(e)$ denote the normal form of an expression $e \in \mathrm{Reg}_X^\vee$ in the following convergent term rewriting system:

$$(a+b)^\vee \to a^\vee + b^\vee \qquad \mathbb{0}^\vee \to \mathbb{0} \qquad (a^\star)^\vee \to (a^\vee)^\star$$

$$(a \cdot b)^\vee \to b^\vee \cdot a^\vee \qquad \mathbb{1}^\vee \to \mathbb{1} \qquad a^{\vee\vee} \to a$$

The corresponding equations being derivable in KAC^-, one easily obtains that:

$$\forall e \in \mathrm{Reg}_X^\vee,\ \mathrm{KAC}^- \vdash \tau(e) = e \tag{8}$$

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

We finally denote by $\mathbf{e}$ the expression obtained by further applying the substitution $\nu \triangleq \left[x^\vee \mapsto x', (\forall x \in \mathbf{X})\right]$, i.e., $\mathbf{e} \triangleq \nu(\tau(e))$. (Note that $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$: it is regular, all occurrences of the converse operation having been eliminated.)

As explained in the introduction, Bloom et al.'s proof [4] amounts to proving the implications (3) and (4). We include a syntactic and elementary presentation of this proof, for the sake of completeness.

Lemma 3. *For all* $e, f \in \mathrm{Reg}_X^\vee$, $e \equiv_{\mathrm{Lang}^\vee} f$ *entails* $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$.

Proof. Let us write $X_\bullet \triangleq X \uplus \{\bullet\}$ and consider the following interpretations (which appear in [4, proof of Proposition 4.3]):

$$\begin{array}{ll} \mu : X \longrightarrow \mathcal{P}\left(X_\bullet^\star\right) & \eta : \mathbf{X} \longrightarrow \mathcal{P}\left(X_\bullet^\star\right) \\ \quad x \longmapsto \{x \cdot \bullet\} & \quad x \in X \longmapsto \{x \cdot \bullet\} \\ & \quad x' \in X' \longmapsto \{\bullet \cdot x\} \end{array}$$

One can check that $\widehat{\eta}$ is injective modulo equality of denoted languages (see Appendix A), in the sense that for any expression $e \in \mathrm{Reg}_{\mathbf{X}}$, we have

$$\widehat{\eta}(e) = \widehat{\eta}(f) \text{ implies that } [\![e]\!] = [\![f]\!] \ . \tag{9}$$

By a simple induction on e, we get $\widehat{\mu}(\tau(e)) = \widehat{\eta}(\nu(\tau(e))) = \widehat{\eta}(\mathbf{e})$. Using (8), we further obtain that $\tau(e) \equiv_{\mathrm{Lang}^\vee} e$. We thus deduce that $\widehat{\mu}(e) = \widehat{\eta}(\mathbf{e})$. All in all, we obtain: $e \equiv_{\mathrm{Lang}^\vee} f \Rightarrow \widehat{\mu}(e) = \widehat{\mu}(f) \Rightarrow \widehat{\eta}(\mathbf{e}) = \widehat{\eta}(\mathbf{f}) \Rightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$. □

The second implication is even more immediate, using the completeness of KA with respect to language equivalence.

Lemma 4. *For all* $e, f \in \mathrm{Reg}_X^\vee$, *if* $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$ *then* $\mathrm{KAC}^- \vdash e = f$.

Proof. By completeness of KA [7,10], if $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$, then we know that there is a proof $\pi_1 : \mathrm{KA} \vdash \mathbf{e} = \mathbf{f}$. As KA is contained in KAC^-, the same proof can be seen as $\pi_1 : \mathrm{KAC}^- \vdash \mathbf{e} = \mathbf{f}$. By substituting x' by $(x^\vee)$ everywhere in this proof, we get a new proof $\pi_2 : \mathrm{KAC}^- \vdash \tau(e) = \tau(f)$. By (8) and transitivity we thus get $\mathrm{KAC}^- \vdash e = f$. □

We finally deduce that $e \equiv_{\mathrm{Lang}^\vee} f \Leftrightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \Leftrightarrow \mathrm{KAC}^- \vdash e = f$. Since the regular expressions $\mathbf{e}$ and $\mathbf{f}$ can be easily computed from e and f, the problem of equivalence in KAC^- thus reduces to an equality of regular languages, which makes it decidable.

2.2. Relations with converse: theory KAC

We now move to the equational theory generated by relational models. It turns out that this theory is characterised using "closed" languages on the extended alphabet $\mathbf{X}$. To define this closure operation, we first define a mirror operation $\overline{w}$ on words over $\mathbf{X}$, such that $\overline{\epsilon} \triangleq \epsilon$ and for any $x \in \mathbf{X}$ and $w \in \mathbf{X}^\star$, $\overline{wx} \triangleq x'\overline{w}$. Accordingly with the axiom (5) of KAC we define a reduction relation $\rightsquigarrow$ on words over $\mathbf{X}$, using the following word rewriting rule

$$w\overline{w}w \rightsquigarrow w \tag{10}$$

We call $w\overline{w}w$ a *pattern* of *root* w. The *last two thirds* of the pattern are $\overline{w}w$. Following [4,5], we extend this relation into a closure operation on languages.

Definition 5. The *closure* of a language $L \subseteq \mathbf{X}^\star$ is the smallest language containing L that is downward-closed with respect to $\rightsquigarrow$:

$$cl(L) \triangleq \left\{ v \mid \exists u \in L : u \rightsquigarrow^\star v \right\} \ .$$

Example 6. If $X = \{a, b, c, d\}$, then $\mathbf{X} = \left\{a, b, c, d, a', b', c', d'\right\}$, and $\overline{ab'} = ba'$. We have the reduction $cab'ba'ab'd' \rightsquigarrow cab'd'$, by triggering a pattern of root ab'. For $L = \left\{aa'a, b, cab'ba'ab'd'\right\}$, we have $cl(L) = L \cup \left\{a, cab'd'\right\}$.

Now we define a family of languages which play a prominent role in the sequel.

Definition 7. For any word $w \in \mathbf{X}^\star$, we define a regular language $\Gamma(w)$ by:

$$\begin{array}{ll} & \Gamma(\epsilon) \triangleq \{\epsilon\} \\ \forall x \in \mathbf{X}, \forall w \in \mathbf{X}^\star, & \Gamma(wx) \triangleq \left(x'\Gamma(w)x\right)^\star \ . \end{array}$$

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

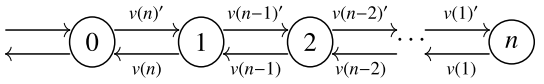


Fig. 1. Automaton $\mathscr{G}(v)$ recognising $\Gamma(v)$, with $|v| = n$.

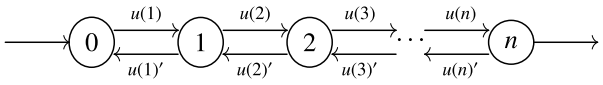


Fig. 2. Automaton $\Phi(u)$, with $|u| = n$.

An equivalent operator called G is used in [4]: we actually have $\Gamma(w) = G(\overline{w})$, and our recursive definition directly corresponds to [4, Proposition 5.11.(2)]. By using such a simple recursive definition, we avoid the need for the notion of *admissible maps*, which is extensively used in [4].

Instead, we just have the following property to establish, which illustrates why these languages are of interest: words in $\Gamma(w)$ reduce into the last two thirds of a pattern compatible with w. Therefore, in the context of recognition by an automaton, $\Gamma(w)$ contains all the words that could potentially be skipped after reading w, in a closure automaton.

Proposition 8. *For all words u and v, $u \in \Gamma(v) \Leftrightarrow \exists t \in \mathrm{suffixes}(v) : u \rightsquigarrow^\star \overline{t}t$.*

Proof. The proof of the implication from left to right is routine but a bit lengthy, we put it in Appendix B.

For the converse implication, we first define the following language: $\Gamma'(v) \triangleq \{\overline{t}t \mid t \in \mathrm{suffixes}(v)\}$. We thus have to show that the upward closure of $\Gamma'(v)$ is contained in $\Gamma(v)$. We first check that this language satisfies $\Gamma'(\epsilon) = \epsilon$ and $\Gamma'(vx) = \epsilon + x'\Gamma'(v)x$, which allows us to deduce that $\Gamma'(v) \subseteq \Gamma(v)$ by a straightforward induction.

It thus suffices to show that $\Gamma(v)$ is upward-closed with respect to $\rightsquigarrow$. For this, we introduce the family of automata $\mathscr{G}(v)$ depicted in Fig. 1. One can check that $\mathscr{G}(v)$ recognises $\Gamma(v)$ by a simple induction on v. One can moreover notice that in this automaton, if $p \xrightarrow{x}_{\mathscr{G}(v)} q$, then $q \xrightarrow{x'}_{\mathscr{G}(v)} p$. More generally, for any word u, if $p \xrightarrow{u}_{\mathscr{G}(v)} q$, then $q \xrightarrow{\overline{u}}_{\mathscr{G}(v)} p$. So if $u_1 w u_2 \in \Gamma(v)$, then by definition of the automaton we have $0 \xrightarrow{u_1}_{\mathscr{G}(v)} q_1 \xrightarrow{w}_{\mathscr{G}(v)} q_2 \xrightarrow{u_2}_{\mathscr{G}(v)} 0$, and thus, by the previous remark:

$$0 \xrightarrow[\mathscr{G}(v)]{u_1} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{\overline{w}} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{u_2} 0 \quad ,$$

i.e., $u_1 w \overline{w} w u_2 \in \Gamma(v)$. In other words, for any words v and w and any $u \in \Gamma(v)$, if $w \rightsquigarrow u$ then w is also in $\Gamma(v)$, meaning exactly that $\Gamma(v)$ is upward-closed with respect to $\rightsquigarrow$.

Since $\Gamma'(v) \subseteq \Gamma(v)$, we deduce that $\Gamma(v)$ contains the upward closure of $\Gamma'(v)$, as expected. □

We now have enough material to embark in the proof of the implication (6) from the introduction, stating that if two expressions $e, f \in \mathrm{Reg}^\vee_X$ are equal for all interpretations in all relational models, then $cl([\![\mathbf{e}]\!]) = cl([\![\mathbf{f}]\!])$.

Proof. Bloom et al. [4] consider specific relational interpretations: for any word $u \in \mathbf{X}^\star$ and for any letter $x \in \mathbf{X}$, they define

$$\phi_u(x) \triangleq \{(i-1, i) \mid u(i) = x\} \cup \{(i, i-1) \mid u(i) = x'\} \subseteq \{0, \dots, n\}^2 \quad ,$$

where $n \triangleq |u|$. The key property of those interpretations is the following:

$$(0, n) \in \widehat{\phi_u}(v) \Leftrightarrow v \rightsquigarrow^\star u \quad . \tag{11}$$

We give a new proof of this property, by using the automaton $\Phi(u)$ depicted in Fig. 2. By definition of $\Phi(u)$ and ϕ_u, we have that

$$(i, j) \in \phi_u(x) \Leftrightarrow i \xrightarrow{x}_{\Phi(u)} j \quad .$$

Therefore, proving (11) amounts to proving

$$v \in L(\Phi(u)) \Leftrightarrow v \rightsquigarrow^\star u \quad . \tag{12}$$

First notice that $i \xrightarrow{x}_{\Phi(u)} j \Leftrightarrow j \xrightarrow{x'}_{\Phi(u)} i$. We extend this to paths (as in the proof of Proposition 8) and then prove that if $s \rightsquigarrow t$ and $i \xrightarrow{t}_{\Phi(u)} j$ then $i \xrightarrow{s}_{\Phi(u)} j$. As u is clearly in $L(\Phi(u))$, any v such that $v \rightsquigarrow^\star u$ is also in $L(\Phi(u))$.

We proceed by induction on u for the other implication. The case $u = \epsilon$ being trivial, we consider $v \in L(\Phi(xu))$. We introduce a second automaton $\Phi'(xu)$ given in Fig. 3, that recognises the same language as $\Phi(xu)$. The upper part of this automaton is actually the automaton $\mathscr{G}(\overline{xu})$ (as given in Fig. 1), recognising the language $\Gamma(\overline{xu})$. Moreover, the lower

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

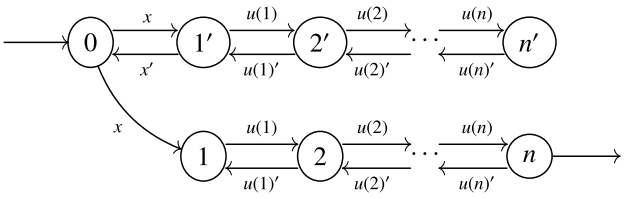


Fig. 3. Automaton $\Phi'(xu)$, with $|u| = n$, language equivalent to $\Phi(xu)$.

part starting from state 1 is the automaton $\Phi(u)$. This allows us to obtain that $L(\Phi(xu)) = \Gamma(\overline{xu})xL(\Phi(u))$. Hence, for any $v \in L(\Phi(xu))$, there are $v_1 \in \Gamma(\overline{xu})$ and $v_2 \in L(\Phi(u))$ such that $v = v_1 x v_2$. By induction, we get $v_2 \rightsquigarrow^\star u$, and by Proposition 8 we know that $v_1 \rightsquigarrow^\star \overline{w}w$, with $w \in \text{suffixes}(\overline{xu})$. That means that $\overline{xu} = tw$, for some word t, so $xu = \overline{tw} = \overline{w}\,\overline{t}$. If we put everything back together:

$$v = v_1 x v_2 \rightsquigarrow^\star v_1 xu \rightsquigarrow^\star \overline{w}wxu = \overline{w}w\overline{w}\,\overline{t} \rightsquigarrow \overline{w}\,\overline{t} = xu \quad .$$

This concludes the proof of (12), and thus (11).

We follow Bloom et al.'s proof [4] to deduce that the implication (6) from the introduction holds: we first prove that for all $e \in \text{Reg}_{\mathbf{X}}$, we have

$$\begin{aligned} u \in cl([\![e]\!]) &\Leftrightarrow \exists v \in [\![e]\!],\, v \rightsquigarrow^\star u && \text{(by definition)} \\ &\Leftrightarrow \exists v \in [\![e]\!],\, (0,n) \in \widehat{\phi_u}(v) && \text{(by (11))} \\ &\Leftrightarrow (0,n) \in \widehat{\phi_u}(e) \quad . \end{aligned}$$

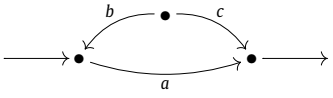
(For the last line, we use the fact that for any relational interpretation ϕ, we have $\widehat{\phi}(e) = \bigcup_{v \in [\![e]\!]} \widehat{\phi}(v)$.)

Furthermore, as $\phi_u(x') = \phi_u(x)^\vee$, we can prove that $\widehat{\phi_u}(e) = \widehat{\phi_u}(\mathbf{e})$. Therefore, for all expressions $e, f \in \text{Reg}_X^\vee$ such that $e \equiv_{\text{Rel}^\vee} f$, we have $\widehat{\phi_u}(\mathbf{e}) = \widehat{\phi_u}(e) = \widehat{\phi_u}(f) = \widehat{\phi_u}(\mathbf{f})$, and we deduce that $cl([\![\mathbf{e}]\!]) = cl([\![\mathbf{f}]\!])$ thanks to the above characterisation. □

2.3. A connection with subpositive relation algebra

The above notion of reduction ($\rightsquigarrow^\star$) can be characterised as an instance of a more general notion which appear in the (apparently independent) works of Freyd and Scedrov [12] and Andréka and Bredikhin [13] on subpositive algebras of relations. Although we do not need it in the sequel, we describe this connection here, to shed some light on the relationships between those works and Bloom et al.'s results.

Subpositive algebras, that correspond to representable allegories in Freyd and Scedrov's terminology, are relations algebras over the restricted signature $(\cdot, \cap, {}^\vee, \mathbb{1})$: composition, intersection, converse, and identity. Terms over this signature and with variables in X can be represented by 2-pointed directed graphs, whose edges are labelled in X. For instance, the expression $a \cap (b^\vee \cdot c)$ can be represented by the following graph, where the two special nodes are respectively marked with an ingoing and an outgoing unlabelled arrow:



Homomorphisms between such graphs are defined as directed labelled graph homomorphisms that preserve the two pointed nodes. Let us write $G \triangleleft G'$ when there exists such a homomorphism from a graph G' to a graph G.

An important result is that the corresponding equational theory is decidable: denote by $G(u)$ the graph associated to a term u, and by $u \subseteq_{\mathcal{R}el} v$ the inclusion of two terms u, v under all relational interpretations, we have:

Theorem 9. *(See Theorem 1 from [13], or page 208 in [12].) For all terms u, v of subpositive algebras, we have $u \subseteq_{\mathcal{R}el} v$ if and only if $G(u) \triangleleft G(v)$.*

The key step to prove the above theorem consists in relating relational interpretations to graphs. For an interpretation σ from the variables in X into relations on a given set I, denote by $\langle G(\sigma), i, j\rangle$ the graph with set of vertices I, labelled edges $\{(i, x, j) \mid (i, j) \in \sigma(x)\}$, and pointed elements i and j.

Lemma 10. *(See Lemma 3 from [13], or page 208 in [12].) Let u be a term with variables in X, and $\sigma : X \to \mathcal{P}(I \times I)$ a relational interpretation, we have $(i, j) \in \widehat{\sigma}(u)$ if and only if $\langle G(\sigma), i, j\rangle \triangleleft G(u)$.*

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

Now let us restrict to terms without intersection. The associated graphs are isomorphic to words over $\mathbf{X}$. Using (11) and Lemma 10, and noticing that $\langle G(\phi_u), 0, n\rangle = G(u)$, we thus obtain that for all words $u, v \in \mathbf{X}^\star$,

$$v \rightsquigarrow^\star u \;\Leftrightarrow\; G(u) \lhd G(v) \;. \tag{13}$$

This is consistent with [4]: homomorphisms between such linear graphs are precisely what Bloom et al. define as the set of *admissible functions* γ from the prefixes of the word v to the prefixes of u such that $\gamma(v) = u$ and they show that $v \rightsquigarrow^* u$ if and only if there exists such a map [4, Proposition 5.13].

3. Confluence of the reduction relation

When considering $\rightsquigarrow$ as a rewriting relation, we wondered whether it is confluent or not. It turns out that it is confluent. This property is not required for the proofs to come, but we include the result here for the sake of completeness.

First, notice that the irreflexive part of this relation is terminating: either $|uw\overline{w}wv| > |uwv|$, or $w = \epsilon$ and the two words are equal. By Newman's lemma [16], it thus suffices to establish local confluence:

Lemma 11 *(Local confluence). Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rightsquigarrow m_1$ and $m \rightsquigarrow m_2$. There exists $n \in \mathbf{X}^\star$ such that $m_1 \rightsquigarrow^\star n$ and $m_2 \rightsquigarrow^\star n$.*

As we were trying to prove this lemma, it became obvious that the proof would be long and repetitive: the only strategy we could find was an exhaustive study of all critical pairs, and there are many of them. We thus used the proof assistant Coq to help us in that task:

- the statement of the lemma is short and involves very few notions, so that the encoding of the problem in Coq was immediate;
- this allowed us to get assurance that no critical pair was overlooked, something difficult to achieve with a pen-and-paper case analysis: there are thirty-five key cases, and this disjunction is not trivial to establish;
- we were able to mechanise a significant part of the proof, using the tactic language of Coq to implement pattern recognition and automatic reduction. For instance, in a case like

$$\begin{cases} m = u_1 w_1 \overline{w_1} w_1 v_1 \\ w_1 = x w_2 \overline{w_2} w_2 y \\ m_1 = u_1 x w_2 \overline{w_2} w_2 y v_1 \\ m_2 = u_1 w_1 \overline{w_1} x w_2 y u_1, \end{cases}$$

  one only needs to recognise the patterns in m_1 and m_2 and reduce them as much as possible using relation $\rightsquigarrow$ to get $n = u_1 x w_2 y v_1$.

Then the proof is just an exploration of the possible sub-cases, with the automatic tactic dealing swiftly with the administrative cases, and leaving only the subtle cases to be explicitly proven. Over the thirty-five cases that arise naturally, seventeen are ruled-out by using size considerations, and twelve are mere questions of rewriting that were solved automatically. All in all, we only had to solve six sub-cases by hand, by using appropriate pumping lemmas.

By instrumenting the proof script to keep track of the uses of the relation $\rightsquigarrow$, we were actually able to establish a slightly stronger result: four steps suffice to join all critical pairs.

Lemma 12. *Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rightsquigarrow m_1$ and $m \rightsquigarrow m_2$. There exists $n \in \mathbf{X}^*$ and $k_1, k_2 \leqslant 4$ such that $m_1 \rightsquigarrow^{k_1} n$ and $m_2 \rightsquigarrow^{k_2} n$.*

Corollary 13 *(Confluence). Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rightsquigarrow^\star m_1$ and $m \rightsquigarrow^\star m_2$. There exists $n \in \mathbf{X}^*$ such that $m_1 \rightsquigarrow^\star n$ and $m_2 \rightsquigarrow^\star n$.*

The Coq proof is available online [17]; note that the converse of the reduction relation $\rightsquigarrow$ is also confluent, albeit not terminating; this is much easier to prove.

4. Closure of an automaton

The problem here is the following: given two regular expressions $e, f \in \mathrm{Reg}^\vee_X$, how to decide $cl(\llbracket \mathbf{e} \rrbracket) = cl(\llbracket \mathbf{f} \rrbracket)$? We follow the approach proposed by Bloom et al.: given an automaton recognising a language L, we show how to construct an automaton recognising $cl(L)$. To solve the initial problem, it then suffices to build two automata recognising $\llbracket \mathbf{e} \rrbracket$ and $\llbracket \mathbf{f} \rrbracket$, to apply a construction to obtain two automata for $cl(\llbracket \mathbf{e} \rrbracket)$ and $cl(\llbracket \mathbf{f} \rrbracket)$, and to check those for language equivalence.

As a starting point, we first recall the construction proposed in [4].

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

4.1. Original construction

This construction uses the transition monoid of the input automaton:

Definition 14 *(Transition monoid).* Let $\mathscr{A} = \langle Q, \Sigma, q_0, \mathscr{T}, \delta\rangle$ be a deterministic automaton. Each word $u \in \Sigma^\star$ induces a function $u_\mathscr{A} : Q \to Q$ which associates to a state p the state q obtained by following the unique path from p labelled by u. The *transition monoid* of $\mathscr{A}$, written $M_\mathscr{A}$, is the set of functions $Q \to Q$ induced by words of $\Sigma^\star$, equipped with the composition of functions and the identity function.

This monoid is finite, and its subsets form a Kleene Algebra. Bloom et al. then proceed to define the closure automaton in the following way:

Theorem 15 *(Closure automaton). (See [4].) Let $L \subseteq \mathbf{X}^\star$ be a regular language, recognised by the deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, q_0, Q_f, \delta\rangle$. Let $M_\mathscr{A}$ be the transition monoid of $\mathscr{A}$. Then the following deterministic automaton recognises $cl(L)$:*

$$\begin{aligned} \mathscr{B} &\triangleq \langle \mathcal{P}(M_\mathscr{A}) \times \mathcal{P}(M_\mathscr{A}), \mathbf{X}, (\{\epsilon_\mathscr{A}\}, \{\epsilon_\mathscr{A}\}), \mathscr{T}, \delta_1\rangle \\ \text{with } \mathscr{T} &\triangleq \left\{(F,G) \;\middle|\; \exists u_\mathscr{A} \in F : u_\mathscr{A}(q_0) \in Q_f\right\}, \\ \text{and } \delta_1((F,G),x) &\triangleq \left(F \cdot \{x_\mathscr{A}\} \cdot \left(\left(\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\}\right)^\star\right),\ \left(\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\}\right)^\star\right). \end{aligned}$$

An important idea in this construction, which leads to the presented one, is the transition rule for the second component above. Let us write $\delta_2(G,x)$ for the expression $\left(\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\}\right)^\star$, so that the definition of δ_1 can be reformulated as

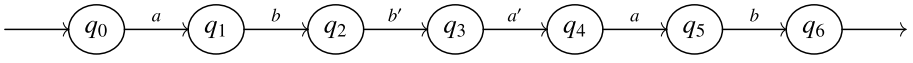
$$\delta_1((F,G),x) = (F \cdot \{x_\mathscr{A}\} \cdot \delta_2(G,x),\ \delta_2(G,x)).$$

With that in mind, one can see the second component as some kind of *history*, that runs on its own, and is used at each step to enrich the first component. At this point, it might be interesting to notice that the formula for $\delta_2(G,x)$ closely resembles the one for $\Gamma(wx) = \left(x'\Gamma(w)x\right)^\star$, which we defined in Section 2.2.

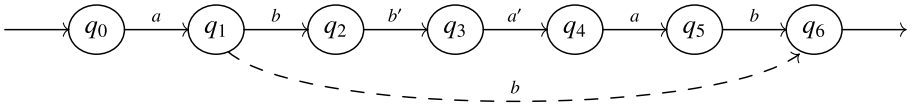
4.2. Intuitions

Let us try to build a closure automaton. One way would be to simply add transitions to the initial automaton. This idea comes naturally when one realises that if $u \rightsquigarrow^\star v$, then v is obtained by erasing some subwords from u: at each reduction step $u_1 w\overline{w}wu_2 \rightsquigarrow u_1 wu_2$ we just erase $\overline{w}w$. To "erase" such subwords using an automaton, it suffices to allow one to jump along certain paths.

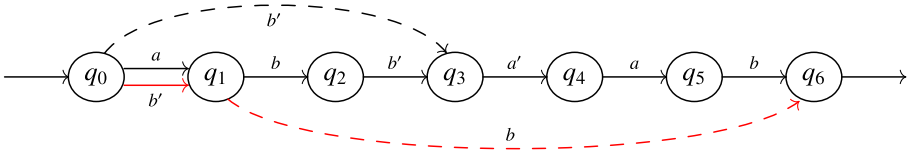
Suppose for instance that we start from the following automaton:



We can detect the pattern $ab\overline{ab}ab$, and allow one to "jump" over it when reading the last letter of the root of the pattern, in this case the b in second position. The automaton thus becomes:



However, this approach is too naive, and it quickly leads to errors. If for instance we slightly modify the above example by adding a transition labelled by b' between q_0 and q_1, the same method leads to the following automaton, by detecting the patterns $b'bb'$ between q_0 and q_3 and $abb'a'ab$ between q_0 and q_6.



The problem is that the word $b'b$ is now wrongly recognised in the produced automaton. What happens here is that we can use the jump from q_1 to q_6, even though we did not read the prerequisite for doing so, in this case the a constituting the beginning of the root ab of pattern $ab\overline{ab}ab$. (Note that the dual idea, consisting in enabling a jump when reading the first letter of the root of the pattern, would lead to similar problems.)

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

A way to prevent that, which was implicitly introduced in the original construction, consists in using a notion of *history*. The states of the closure automaton will be pairs of a state in the initial automaton and a history. That will allow us to distinguish between the state q_1 after reading a and the state q_1 after reading b', and to specify which jumps are possible considering what has been previously read. In the construction given in [4], the history is given by an element of $\mathcal{P}(M_{\mathscr{A}})$, in the second component of the states (the "G" part). We will define a history as a relation over states containing the jumps we are allowed to make after having read some word w, using $\Gamma(w)$.

4.3. New construction

We have shown in Section 2.2 that $\forall u \in \Gamma(w), \exists v \in \text{suffixes}(w) : u \rightsquigarrow^\star \overline{v}v$, so we do have a characterisation of the words "we are allowed to jump over" after having read some word w. The problem is that we want a finite number of possible histories, and there are infinitely many $\Gamma(w)$ (for instance, all the $\Gamma(a^n)$ are different). To get that, we will project $\Gamma(w)$ on the automaton. Let us consider a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, \mathscr{T}, \Delta\rangle$ recognising a language L.

Definition 16. Let $\mathcal{G}(Q)$ be the set of reflexive transitive binary relations over states of $\mathscr{A}$. Given a letter $x \in \mathbf{X}$, we denote by h_x the following increasing function on binary relations between states of $\mathscr{A}$.

$$\begin{aligned} h_x &: \mathcal{G}(Q) \to \mathcal{G}(Q) \\ R &\mapsto \big(\Delta(x') \cdot R \cdot \Delta(x)\big)^\star \end{aligned}$$

For any word $w \in \mathbf{X}^\star$ we call *history* of the word w the relation $[w]$ defined inductively by $[\epsilon] \triangleq \mathrm{Id}_Q$ and $[wx] \triangleq h_x([w])$.

One can notice right away the strong relationship between $[\cdot]$ and Γ:

Proposition 17. $\forall w, q_1, q_2,\ (q_1, q_2) \in [w] \Leftrightarrow \exists u \in \Gamma(w) : q_1 \xrightarrow{u}_{\mathscr{A}} q_2$.

This result is straightforward once one realises that $[w] = \widehat{\sigma}(\Gamma(w))$ with $\sigma(x) = \Delta(x)$. By composing Propositions 8 and 17 we eventually obtain that $(q_1, q_2) \in [w]$ if and only if $\exists u : q_1 \xrightarrow{u}_{\mathscr{A}} q_2$ and $u \rightsquigarrow^\star \overline{v}v$, with v a suffix of w.

We now have all the tools required to define an automaton for the closure of $\mathscr{A}$:

Theorem 18 *(Closure automaton). The closure of the language L is recognised by the automaton*

$$\begin{aligned} &\mathscr{A}' \triangleq \langle Q \times \mathcal{G}(Q), \mathbf{X}, I \times \{\mathrm{Id}_Q\}, \mathscr{T} \times \mathcal{G}(Q), \Delta'\rangle\ , \\ &\textit{where}\quad \Delta' \triangleq \{((q_1, R), x, (q_2, h_x(R))) \mid (q_1, q_2) \in \Delta(x) \cdot h_x(R)\}\ . \end{aligned}$$

We shall write L' for the language recognised by $\mathscr{A}'$. One can read the set of transitions as "from a state q_1 with a history R, perform a step x in the automaton $\mathscr{A}$, and then a jump compatible with $h_x(R)$, which becomes the new history". An example of this construction is given in Section 4.4. It is useful to notice at this point that if (p, R) is an accessible state in $\mathscr{A}'$, then there is some word u such that $(i, \mathrm{Id}_Q) = (i, [\epsilon]) \xrightarrow{u}_{\mathscr{A}'} (p, R)$, from which we can deduce by induction on u that $R = [u]$. One can see, from the definition of Δ' and Proposition 17 that:

$$\exists (q_2, v) \in Q \times \Gamma(ux) : q_1 \xrightarrow{x}_{\mathscr{A}} q_2 \xrightarrow{v}_{\mathscr{A}} q_3 \Leftrightarrow (q_1, [u]) \xrightarrow{x}_{\mathscr{A}'} (q_3, [ux]). \tag{14}$$

Now we prove the correctness of this construction. First recall the notion of *simulation* [18]:

Definition 19 *(Simulation).* A relation S between the states of two automata $\mathscr{A}$ and $\mathscr{B}$ is a *simulation* if for all $p\ S\ q$ we have (a) if $p \xrightarrow{x}_{\mathscr{A}} p'$, then there exists q' such that $q \xrightarrow{x}_{\mathscr{B}} q'$ and $p'\ S\ q'$, and (b) if $p \in \mathscr{T}_{\mathscr{A}}$ then $q \in \mathscr{T}_{\mathscr{B}}$. We say that $\mathscr{A}$ *is simulated by* $\mathscr{B}$ if there is a simulation S such that for any $p_0 \in I_{\mathscr{A}}$, there is $q_0 \in I_{\mathscr{B}}$ such that $p_0\ S\ q_0$.

The following property of $[\cdot]$ is proved by exhibiting such a simulation:

Proposition 20. *For all words $u, v \in \mathbf{X}^\star$ such that $u \rightsquigarrow v$, we have $[u] \subseteq [v]$.*

Proof. First, notice that $\Gamma(u) \subseteq \Gamma(v) \Rightarrow [u] \subseteq [v]$, using Proposition 17. It thus suffices to prove $u \rightsquigarrow v \Rightarrow \Gamma(u) \subseteq \Gamma(v)$, which can be rewritten as $\Gamma(u_1 w \overline{w} w u_2) \subseteq \Gamma(u_1 w u_2)$. We can drop u_2 (it is clear that $\Gamma(w_1) \subseteq \Gamma(w_2) \Rightarrow \forall x \in \mathbf{X}, \Gamma(w_1 x) \subseteq \Gamma(w_2 x)$, from the definition of Γ): we now have to prove that $\Gamma(u_1 w \overline{w} w) \subseteq \Gamma(u_1 w)$. The proof of this inclusion relies on the fact that the automaton $\mathscr{G}(u_1 w \overline{w} w)$ is simulated by the automaton $\mathscr{G}(u_1 w)$.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

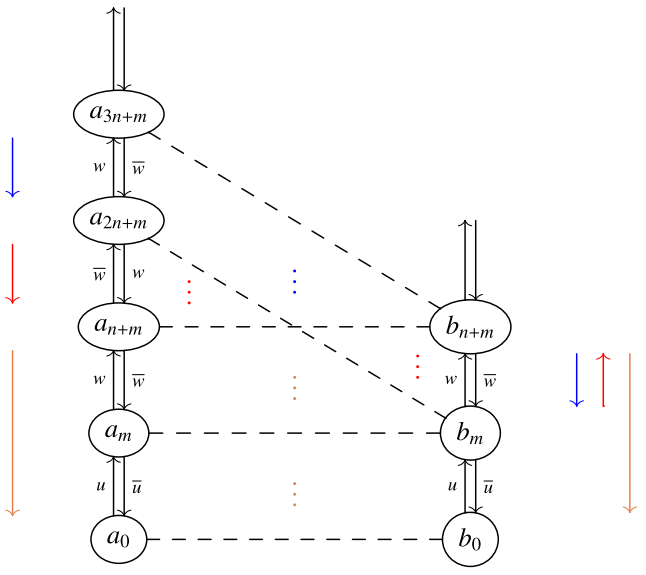


Fig. 4. Automata $\mathscr{G}(uw\overline{w}w)$ and $\mathscr{G}(uw)$, with $|u| = m$ and $|w| = n$.

First, we give in Fig. 4 an abstract view of the automata recognising $\Gamma(uw\overline{w}w)$ and $\Gamma(uw)$ defined as before. With the notations of this figure, now define a relation $\leqslant$ as follows (this relation is also represented in dashed lines in Fig. 4):

$$\begin{aligned} a_i &\leqslant b_i && \text{for all } i \leq n+m \ , \\ a_{n+m+i} &\leqslant b_{n+m-i} && \text{for all } i \leq n \ , \\ a_{2n+m+i} &\leqslant b_{m+i} && \text{for all } i \leq n \ ; \end{aligned}$$

One easily checks that this relation is a simulation, thus establishing in particular that the language recognised by the left-hand side automaton (for $\Gamma(uw\overline{w}w)$) is contained in that of the right-hand side (for $\Gamma(uw)$). □

We define an order relation $\preccurlyeq$ on the states of the produced automaton $(Q \times \mathcal{G}(Q))$, by $(p, R) \preccurlyeq (q, S)$ when $p = q$ and $R \subseteq S$.

Proposition 21. *The relation $\preccurlyeq$ is a simulation for the automaton $\mathscr{A}'$.*

Proof. Suppose that $(p, R) \preccurlyeq (q, S)$ and $(p, R) \xrightarrow{x}_{\mathscr{A}'} (p', h_x(R))$, i.e., $(p, p') \in \Delta(x) \cdot h_x(R)$. We have $p = q$ and $R \subseteq S$, hence $h_x(R) \subseteq h_x(S)$, and thus $(p, p') \in \Delta(x) \cdot h_x(S)$ meaning that $(p, S) \xrightarrow{x}_{\mathscr{A}'} (p', h_x(S))$. It remains to check that $(p', h_x(R)) \preccurlyeq (p', h_x(S))$, i.e., $h_x(R) \subseteq h_x(S)$, which we just proved. □

We may now prove that $L' = cl(L)$.

Lemma 22. $L' \subseteq cl(L)$.

Proof. We prove by induction on u that for all q_0, q such that $(q_0, [\epsilon]) \xrightarrow{u}_{\mathscr{A}'} (q, [u])$, there exists v such that $v \rightsquigarrow^\star u$ and $q_0 \xrightarrow{v}_{\mathscr{A}} q$. The case $u = \epsilon$ is trivial.

If $(q_0, [\epsilon]) \xrightarrow{u}_{\mathscr{A}'} (q_1, [u]) \xrightarrow{x}_{\mathscr{A}'} (q, [ux])$, by induction one can find v_1 such that $q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1$ and $v_1 \rightsquigarrow^\star u$. We also know (by (14) and Proposition 8) that there are some q_2, v_2 and $v_3 \in \text{suffixes}(ux)$ such that $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$, $v_2 \rightsquigarrow^\star \overline{v}_3 v_3$ and $q_2 \xrightarrow{v_2}_{\mathscr{A}} q$. We thus get

$$q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1 \xrightarrow{x}_{\mathscr{A}} q_2 \xrightarrow{v_2}_{\mathscr{A}} q \text{ and } v_1 x v_2 \rightsquigarrow^\star u x v_2 \rightsquigarrow^\star u x \overline{v}_3 v_3 \rightsquigarrow ux.$$

By choosing $q \in \mathscr{T}$, we obtain the desired result. □

Lemma 23. $L \subseteq L'$.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

Proof. First notice that for all $R \in \mathcal{G}(Q)$, $h_x(R)$ is a reflexive relation, hence $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$ entails $\forall R, (q_1, R) \xrightarrow{x}_{\mathscr{A}'} (q_2, h_x(R))$. This means that the relation S defined by $p\ S\ (q, R) \Leftrightarrow p = q$ is a simulation between $\mathscr{A}$ and $\mathscr{A}'$, and thus $L = L(\mathscr{A}) \subseteq L(\mathscr{A}') = L'$. □

Lemma 24. *L' is downward-closed for $\rightsquigarrow$.*

A technical lemma is required to establish this closure property:

Lemma 25. *If $(q_1, [uw]) \xrightarrow{x}_{\mathscr{A}'} (q_2, [uwx]) \xrightarrow{\overline{wx}\ wx}_{\mathscr{A}'} (q_3, [uwx\,\overline{wx}\ wx])$, then $(q_1, [uw]) \xrightarrow{x}_{\mathscr{A}'} (q_3, [uwx])$.*

Proof. If $|w| = n$ and $|u| = m$, the premise can be equivalently stated:

$$(q_1, [(uw)|_{m+n-1}]) \xrightarrow{w(n)}_{\mathscr{A}'} (q_2, [uw]) \xrightarrow{\overline{w}w}_{\mathscr{A}'} (q_3, [uw\overline{w}w]).$$

(Recall that $u|_i$ denotes the prefix of length i of a word u.) Let us write $\Gamma_i = \Gamma\left((uw\overline{w}w)|_{m+n+i}\right) = \Gamma\left(uw(\overline{w}w)|_i\right)$ and $x_i = (uw\overline{w}w)(n+m+i)$ for $0 \leqslant i \leqslant 2n$. By Proposition 17 and the definition of $\mathscr{A}'$, we can show that there are $v_i \in \Gamma_i$ such that the execution above can be lifted into an execution in $\mathscr{A}$:

$$q_1 \xrightarrow{x_0 v_0 x_1 v_1 \cdots x_i v_i \cdots x_{2n} v_{2n}}_{\mathscr{A}} q_3.$$

Then one can prove using Proposition 8 that:

$$\forall i, \exists t_i \in \Gamma(uw) : (\overline{w}w)|_i v_i \rightsquigarrow^\star t_i(\overline{w}w)|_i. \tag{15}$$

As v_i is in $\Gamma\left(uw(\overline{w}w)|_i\right)$, we know that there is some suffix t of $uw(\overline{w}w)|_i$ such that $v_i \rightsquigarrow^\star \bar{t}t$. We will do a case analysis on the size of t:

- if $n + i \leqslant |t|$, then there is a suffix s of u such that $t = sw(\overline{w}w)|_i$, so $(\overline{w}w)|_i v_i \rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}\,\overline{w}\ \bar{s}sw(\overline{w}w)|_i$.
  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$ so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i p\bar{s}sw(\overline{w}w)|_i\\ &\rightsquigarrow (\overline{w}w)|_i p\bar{s}sw(\overline{w}w)|_i = \overline{sw}sw(\overline{w}w)|_i.\end{aligned}$$

  - Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_1$ and $w = w_1w_2$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w}w_1\overline{\overline{w}w_1}\,\overline{w}\ \bar{s}sw(\overline{w}w)|_i = \overline{w}_2\ \overline{w}_1w_1\overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w}_2\ \overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i = \overline{w}w\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &\rightsquigarrow \overline{sw}sw(\overline{w}w)|_i.\end{aligned}$$

  As $s \in \text{suffixes}(u)$ we know that $sw \in \text{suffixes}(uw)$, hence $\overline{sw}sw \in \Gamma(uw)$.
- If $i \leqslant |t| < n + i$ then $w = w_1w_2$ and $t = w_2(\overline{w}w)|_i$ so

$$(\overline{w}w)|_i v_i \rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}\,\overline{w_2}w_2(\overline{w}w)|_i$$

  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$. As $\overline{w} = \overline{w_2}\ \overline{w_1}$, we can also compare $(\overline{w}w)|_i$ with $\overline{w_2}$:
    - If $(\overline{w}w)|_i = \overline{w_2}w_3$ then

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w_2}w_3\overline{w_3}w_2\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w_2}w_3\overline{w_3}w_2(\overline{w}w)|_i = (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i\\ &= \overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i}(\overline{w}w)|_i\end{aligned}$$

      And as $\overline{w} = (\overline{w}w)|_i p$, $w = \overline{p}\overline{(\overline{w}w)|_i}$ so $\overline{(\overline{w}w)|_i} \in \text{suffixes}(w) \subseteq \text{suffixes}(uw)$, hence $\overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i} \in \Gamma(uw)$.
    - If on the other hand $\overline{w_2} = (\overline{w}w)|_i w_3$, we have

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i\\ &\rightsquigarrow (\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i = \overline{w_2}w_2(\overline{w}w)|_i\end{aligned}$$

    $w_2 \in \text{suffixes}(w) \subseteq \text{suffixes}(uw)$ so $\overline{w}_2w_2 \in \Gamma(uw)$.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

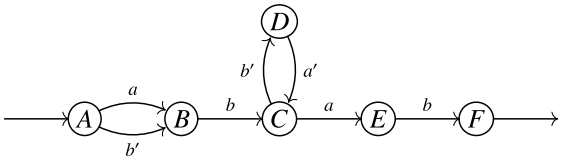


Fig. 5. Initial automaton.

– Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_3$ and $w = w_3w_4$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w}w_3\overline{w_3}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i = \overline{w}w_1w_2\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w}w_1w_2(\overline{w}w)|_i = \overline{w}w(\overline{w}w)|_i\end{aligned}$$

And obviously $\overline{w}w \in \Gamma(uw)$.

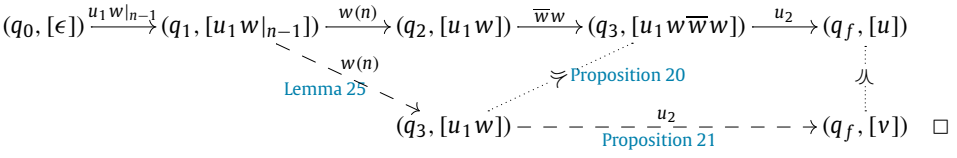
- If $|t| < i$ then $(\overline{w}w)|_i = st$. In this case we have $(\overline{w}w)|_i v_i \rightsquigarrow^\star st\bar{t}t \rightsquigarrow st = \overline{\epsilon}\epsilon(\overline{w}w)|_i$, and $\epsilon \in \text{suffixes}(uw)$ so $\overline{\epsilon}\epsilon \in \Gamma(uw)$.

In all cases, we have shown that $(\overline{w}w)_i v_i \rightsquigarrow^\star t_i(\overline{w}w)|_i$ with $t_i \in \Gamma(uw)$.

We deduce that $v_0x_1v_1\cdots x_iv_i\cdots x_{2n}v_{2n} \rightsquigarrow^\star t_0t_1\cdots t_{2n}\overline{w}w \in \Gamma(uw)^{2n+2} \subseteq \Gamma(uw)$. By Proposition 8, this means that $v_0x_1v_1\cdots x_iv_i\cdots x_{2n}v_{2n}$ is in $\Gamma(uw)$, so that $(q_1, q_3) \in \Delta(w(n)) \cdot [uw]$, and $(q_1, [uw|_{n-1}]) \xrightarrow{w(n)}_{\mathscr{A}'} (q_2, [uw])$. □

With this intermediate lemma, one can obtain a succinct proof of Lemma 24:

Proof. The statement of the lemma is equivalent to saying that if $u \rightsquigarrow v$ with $u \in L'$ then v is also in L'. Consider $u = u_1w \cdot \overline{w} \cdot wu_2$ and $v = u_1wu_2$ with $|w| = n \geqslant 1$ (the case where $w = \epsilon$ does not hold any interest since it implies that $u = v$). By combining Lemma 25, Proposition 20 and Proposition 21 we can build the following diagram:



□

Lemmas 23 and 24 tell us that L' is closed and contains L, so by definition of the closure of a language, we get $cl(L) \subseteq L'$. Lemma 22 gives us the other inclusion, thus proving Theorem 18.

4.4. Example

Let us illustrate this construction on a simple example: consider the automaton depicted in Fig. 5. It recognises the language $[\![(a + b')b(b'a')^\star ab]\!]$, which is not closed. Informally, we observe that

- when starting with an a, and by firing the starred expression only once, a pattern of root ab appears, so that the word ab should belong to the closure. Such a behaviour is no longer possible if we fire the starred expression more than once;
- when starting with a b, and by firing the starred expression at least once, a pattern of root b' appears, so that the language $[\![b'a'(b'a')^\star ab]\!]$ is contained in the closure.

Now the first step to build the closure automaton consists in computing the values of $[\cdot]$; they are summarised in Fig. 6. We can then build the closure automaton as described in Theorem 18. The resulting non-deterministic automaton is drawn in Fig. 7. Due to the history component, the states B and C are duplicated; moreover, "jumps" have been used to obtain the two red transitions, from $(A, [\epsilon])$ to $(D, [b'])$, and from $(B, [a])$ to $(F, [ab])$. Using those transitions, one can notice that the word ab is now accepted, as well as all words from $[\![b'a'(b'a')^\star ab]\!]$.

The determinised version of this automaton is finally given in Fig. 8.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

$$
\begin{array}{lll}
[\,\epsilon\,] = \mathrm{Id}_Q & (= [a'] = [aa'] = [ba'] = [aba'] &) \\
[\,a\,] = \mathrm{Id}_Q \cup \{(D,E)\} & (= [aa] = [b'a] = [ba] = [aba] &) \\
[\,b\,] = \mathrm{Id}_Q \cup \{(A,C)\} & (= [bb] = [b'b] = [b'a'] = [abb] &) \\
[b'] = \mathrm{Id}_Q \cup \{(B,D)\} & (= [ab'] = [bb'] = [b'b'] = [abb']) & \\
[ab] = \mathrm{Id}_Q \cup \{(A,C),(C,F),(A,F)\} & &
\end{array}
$$

Fig. 6. Computation of $[\cdot]$.

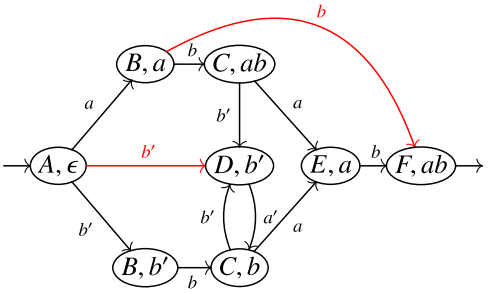


Fig. 7. Closure automaton.

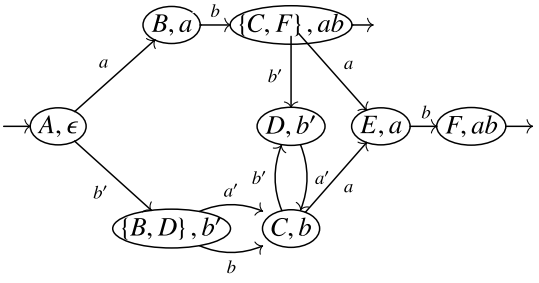


Fig. 8. Determinised version.

5. Analysis and consequences

5.1. Relationship with Bloom et al's construction

As suggested by an anonymous referee, one can also formally relate our construction to the one from [4]: we give below an explicit and rather natural bisimulation relation between the automata produced by both these methods. This results in an alternative correctness proof of the proposed construction, by reducing it to the correctness of the one from [4].

We first make the two constructions comparable: the original construction, because it considers the transition monoid, takes as input a deterministic automaton. It returns a deterministic automaton. Instead, our construction does not require determinism in its input, but produces a non-deterministic automaton. We thus have to ask of both methods to accept as their input a *non-deterministic* automaton, and to return a *deterministic* automaton.

For our construction, the straightforward thing to do would be to determinise the automaton afterwards. We can actually do better, by noticing that from a state $(p,[u])$, reading some letter x, there may be a lot of accessible states, but all of their histories (second components) will be equal to $[ux]$. So in order to get a deterministic automaton, one only has to perform the power-set construction on the first component of the automaton. This way, we get an automaton $\mathscr{A}_1$ with states in $\mathcal{P}(Q) \times \mathcal{G}(Q)$ and a transition function

$$\delta_1((P,R),x) = (P \cdot (\Delta(x) \cdot h_x(R)), h_x(R)).$$

The original construction can also be adjusted quite easily: first build a deterministic automaton $\mathcal{D}$ with the usual powerset construction, then apply the construction as described in Theorem 15 to get an automaton which we call $\mathscr{A}_2$. An important thing here is to understand the shape of the resulting transition monoid $M_{\mathcal{D}}$: its elements are functions over sets of states (because of the power-set construction) induced by words; more precisely, they are sup-semilattice homomorphisms, and they are in bijection with binary relations on states induced by words. (The relation induced by u is the set of pairs (p,q) such that $p \xrightarrow{u} q$.)

Define the following KA-homomorphism from $\mathcal{P}(M_{\mathcal{D}})$ to $\mathcal{P}(Q^2)$:

$$i(F) = \{(p,q) \mid \exists u_{\mathcal{D}} \in F : q \in u_{\mathcal{D}}(\{p\})\}\ .$$

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

(That i is a KA-homomorphism comes from the fact that the elements of $M_{\mathcal{D}}$ are themselves sup-semilattice homomorphisms on $\mathcal{P}(Q)$.) We can check that for all $x \in \mathbf{X}$, we have

$$i(\{x_{\mathcal{D}}\}) = \{(p,q) \mid q \in x_{\mathcal{D}}(\{p\})\} = \{(p,q) \mid q \in \delta(\{p\}, x)\}$$
$$= \left\{(p,q) \;\middle|\; p \xrightarrow{x}_{\mathscr{A}} q\right\} = \Delta(x) \ ,$$

It follows that the following relation is a bisimulation between $\mathscr{A}_1$ and $\mathscr{A}_2$.

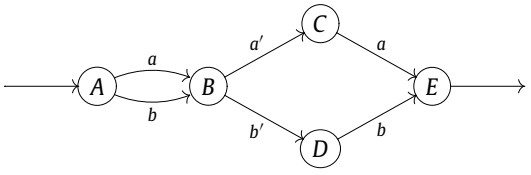
$$\{((I \cdot i(F), i(G)),\ (F,G)) \mid \forall F, G\}$$

We give a detailed proof in Appendix C.

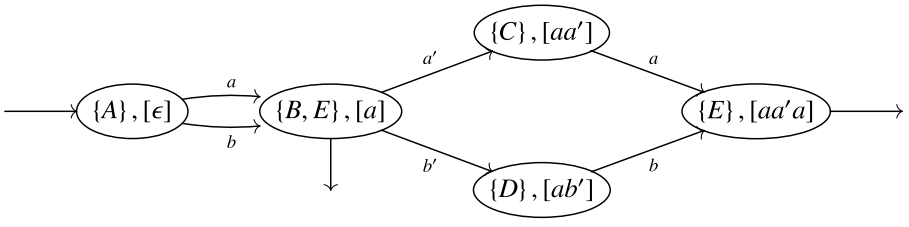
This tight relationship between both constructions may lead to believe the automata built by both constructions are isomorphic. However, it is not the case: the construction presented in this paper produces a smaller automaton than the one given in [4]. Indeed, by unfolding the above bisimulation, one can find a surjective morphism from $\mathscr{A}_2$ to $\mathscr{A}_1$. But such a morphism cannot be found in general in the other direction. This is illustrated in Example 26 below.

Intuitively, a major difference comes from the difference between the determinised automaton and the automaton induced by the right Cayley graph of the transition monoid. In a deterministic automaton, we can define an equivalence relation $u \sim v$, that holds if reading u and reading v in the automaton lead to the same state. In an automaton obtained by a power-set construction, $u \sim v$ is equivalent to saying that for any state p in the original automaton, p can be reached from the initial state by reading u if and only if it can be reached by reading v. In an automaton built by the monoid construction, $u \sim v$ corresponds to saying that for any pair of states p, q in the original automaton, there is a path from p to q labelled by u if and only if there is a path from p to q labelled by v. This second equivalence relation strictly contains the first one.

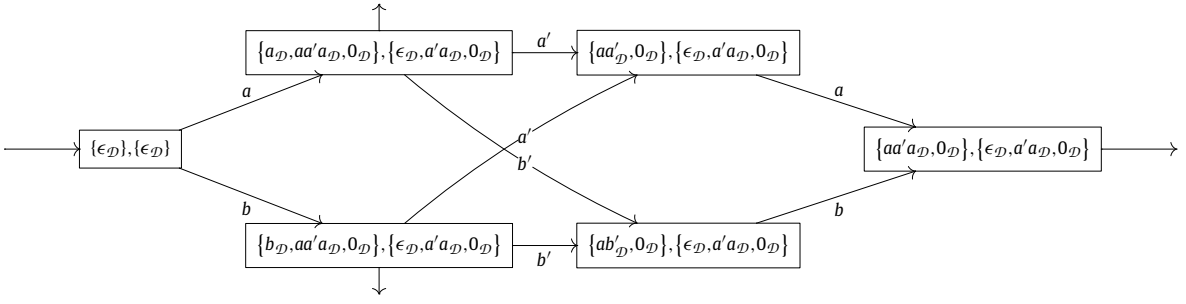
Example 26. Consider the following deterministic automaton $\mathcal{D}$ over the alphabet $\{a, b, a', b'\}$:



By applying the determinised version of our construction, we build the following automaton:



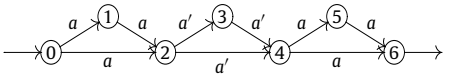
However, the use of the method from [4] give rise to this automaton:



Notice that this automaton is just like the previous one, except that state $\langle\{B, E\}, [a]\rangle$ has been duplicated. This happens because $a_{\mathcal{D}} = \Delta(a) = \{(A,B),(C,E)\} \neq \{(A,B),(D,E)\} = \Delta(b) = b_{\mathcal{D}}$. Thus the monoid will differentiate the state we get after reading a and the state we get after reading b.

Another important difference is the fact that the construction in [4] uses *sets* of relations when we simply use relations. This is particularly visible when looking at the histories. Consider the following automaton $\mathscr{A}$:

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005



The history induced by a in this automaton is $[a] = \left(\Delta(a') \cdot \Delta(a)\right)^\star = \mathrm{Id}_Q \cup \{(2,6);(2,5);(3,5)\}$; and the one induced by aa is $[aa] = \left(\Delta(a') \cdot [a] \cdot \Delta(a)\right)^\star = \mathrm{Id}_Q \cup \{(2,6);(2,5);(3,5)\} = [a]$. However, if we do this with the monoid approach, we can see that $a'a_{\mathscr{A}} = \{(2,6);(2,5);(3,5)\} \neq \{(2,6)\} = a'a'aa_{\mathscr{A}}$. Thus the history we get after a is $\left\{\epsilon_{\mathscr{A}}; a'a_{\mathscr{A}}; 0_{\mathscr{A}}\right\}$ which is a different set that the one we get after aa which is $\left\{\epsilon_{\mathscr{A}}; a'a_{\mathscr{A}}; a'a'aa_{\mathscr{A}}; 0_{\mathscr{A}}\right\}$. This will induce some duplication in the closure automaton. Indeed on this example, our deterministic closure automaton will have 7 states, whereas [4]'s construction produces an automaton with 11 states.

5.2. Complexity

A relevant complexity measure of the final algorithm for deciding equality in KAC is the size of the produced automata. In the following the *size* of an automaton is its number of states. In order to give a fair comparison, we will consider the generic algorithms given in the previous subsection, taking as their input a *non-deterministic* automaton, and returning a *deterministic* automaton.

Let us begin by evaluating the size of the automaton produced by the method in [4], given a non-deterministic automaton of size n. As explained above, the states of the constructed transition monoid ($M_{\mathcal{D}}$) are in bijection with some binary relations on Q. There are thus at most 2^{n^2} elements in this monoid. We deduce that the final automaton, whose states are pairs of subsets of $M_{\mathcal{D}}$ has at most $2^{2^{n^2}} \times 2^{2^{n^2}} = 2^{2^{n^2+1}}$ states.

Now with the deterministic version of our construction, the states are in the set $\mathcal{P}(Q) \times \mathcal{G}(Q)$. Since $\mathcal{G}(Q)$ is the set of reflexive (transitive) relations on Q, we know that $\mathcal{G}(Q)$ has less than $2^{n\times(n-1)}$ elements. Hence we have $|\mathcal{P}(Q) \times \mathcal{G}(Q)| \leqslant 2^n \times 2^{n\times(n-1)} = 2^{n^2}$, which is significantly smaller than the $2^{2^{n^2+1}}$ states we get with the other construction.

5.3. A polynomial-space algorithm

The above upper-bound on the number of states of the automata produced by the presented construction allows us to show that the problem of checking equivalence in KAC is PSpace-complete.

Consider PSpace-hardness first. Using the language-theoretic characterisation from [4,5], KAC is easily shown to be a conservative extension of KA. Indeed, for any regular expression e, we have $\mathbf{e} = e$ and $cl([\![\mathbf{e}]\!]) = [\![e]\!]$, so that for two regular expressions e, f, we have $\mathrm{KAC} \vdash e = f$ iff $cl([\![\mathbf{e}]\!]) = cl([\![\mathbf{f}]\!])$ iff $[\![e]\!] = [\![f]\!]$ iff $\mathrm{KA} \vdash e = f$. This is sufficient to conclude since language equivalence of regular expressions is known to be PSpace-complete [19].

Now recall that the equivalence of two deterministic automata $\mathscr{A}$ and $\mathscr{B}$ is in LogSpace. The algorithm to show that relies on the fact that $\mathscr{A}$ and $\mathscr{B}$ are different if and only if there is a word w in the difference of $L(\mathscr{A})$ and $L(\mathscr{B})$ such that $|w| \leqslant |\mathscr{A}| \times |\mathscr{B}|$. With that in mind, we can give a non-deterministic algorithm, by simulating a computation in both automata with a letter chosen non-deterministically at each step, with a counter to stop us at size $|\mathscr{A}| \times |\mathscr{B}|$. The resulting algorithm will only have to store the counter of size $\log(|\mathscr{A}| \times |\mathscr{B}|)$ and the two current states.

For deciding KAC, the first step is to compute $\mathbf{e}$ and $\mathbf{f}$ from the regular expressions with converse e and f. It is obvious that such a transformation can be done in linear time and space, by a single sweep of both e and f. Then we have to build automata for $\mathbf{e}$ and $\mathbf{f}$. Once again this is a very light operation: if one considers for instance the position automaton (also called Glushkov's construction [20]), we obtain automata of respective sizes $n = |\mathbf{e}| + 1 = |e| + 1$ and $m = |\mathbf{f}| + 1 = |f| + 1$, where $|\cdot|$ denotes the number of variable leaves of a regular expression (possibly with converse).

Our construction then produces closed automata of size at most 2^{n^2} and 2^{m^2}, so that the non-deterministic algorithm to check their equivalence needs to scan all words of size smaller than by $2^{n^2} \times 2^{m^2} = 2^{n^2+m^2}$. The counter used to bound the recursion depth can thus be stored in polynomial space $(n^2 + m^2)$. It is worth mentioning here that with the automata constructed in [4], the counter would have size $2^{n^2+1} + 2^{m^2+1}$ which is not a polynomial.

Now the last two important things to worry about are the representation of the states of the closure automata, in particular their "history" component, and the way to compute their transition function. Let us focus on the automaton for $\mathbf{e}$ and let Q be the set of states of the Glushkov automaton built out of it.

- States are pairs of a set of states in Q and a binary relation (set of pairs) over Q. Such a pair can be stored in polynomial space (recall that $|Q| = n = |e| + 1$).
- For computing the transition function, the image of a pair $(\{q_1, \cdots, q_k\}, R)$ (with $R \subseteq Q^2$) by a letter $x \in \mathbf{X}$ is done in two steps: first the relation becomes $R' = h_x(R) \triangleq \left(\Delta(x') \cdot R \cdot \Delta(x)\right)^\star$, then the set of states becomes $\left\{q \;\middle|\; \exists i, 1 \leqslant i \leqslant k : (q_i, q) \in \Delta(x) \cdot R'\right\}$. Those computations take place in PSpace. (The composition of two relations on Q can be performed in space $\mathcal{O}\left(|Q|^2\right)$, and the same holds for the reflexive and transitive closure of a relation R by building the powers $\left(R + \mathrm{Id}_Q\right)^{2^k}$ and keeping a copy of the previous iteration to stop when the fixed-point is reached.)

Summing up, we obtain Algorithm 1, which is PSpace.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

input : Two regular expressions with converse $e, f \in \mathrm{Reg}_X^\vee$
output: A Boolean, saying whether or not $\mathrm{KAC} \vdash e = f$.

1 $\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, \mathscr{T}_1, \Delta_1 \rangle \leftarrow$ Glushkov's automaton recognising $[\![\mathbf{e}]\!]$;
2 $\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, \mathscr{T}_2, \Delta_2 \rangle \leftarrow$ Glushkov's automaton recognising $[\![\mathbf{f}]\!]$;
3 $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$; /* N gets a value $\geqslant |cl(\mathscr{A}_1)| \cdot |cl(\mathscr{A}_2)|$ */
4 $((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$;
5 **while** $N > 0$ **do**
6 $N \leftarrow N - 1$; /* N bounds the recursion depth */
7 $f_1 \leftarrow$ is_empty$(P_1 \cap \mathscr{T}_1)$;
8 $f_2 \leftarrow$ is_empty$(P_2 \cap \mathscr{T}_2)$;
9 **if** $f_1 = f_2$ **then**
10 $x \leftarrow$random$(\mathbf{X})$; /* Non-deterministic choice */
11 $(R_1, R_2) \leftarrow (h_x(R_1), h_x(R_2))$;
12 $(P_1, P_2) \leftarrow (P_1 \cdot \Delta_1(x) \cdot R_1, P_2 \cdot \Delta_2(x) \cdot R_2)$;
13 **else**
14 **return** false; /* A difference appeared for some word, $e \neq f$ */
15 **end**
16 **end**
17 **return** true; /* There was no difference, $\mathrm{KAC} \vdash e = f$ */

Algorithm 1: A PSPACE algorithm for KAC.

5.4. Time-efficient algorithms

While the previous algorithm matches the theoretical complexity of the problem, it cannot be used in practice. Indeed, it systematically requires exponential time (except when there exists a small counter-example to the starting equation). This is similar to the case of regular expressions without converse (i.e., Kleene algebra), where one usually implements algorithms that are not PSPACE but that require less than exponential time in most cases.

Such algorithms include the standard algorithm by Hopcroft and Karp [21], antichain-based algorithms [22–24], and more recent algorithms relying on "bisimulations up to congruence" [14]. We show how to exploit the latter technique with the automata produced by the presented construction.

To apply this technique, one needs to work with a single non-deterministic automaton, to be determinised by the powerset construction. The global decision procedure for deciding $\mathrm{KAC} \vdash e = f$ still starts by constructing two automata recognising the languages $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$ (as in the first two lines in Algorithm 1). Then one can either

1. consider the disjoint union of their closures,
2. or take their union and build the closure afterwards.

The first approach makes it possible to use the algorithms from [14] off-the-shelf. A drawback is that depending on the constructions used to build the automata for $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$, it might be natural that they share some states (e.g., when using Antimirov's partial derivatives [25]), and we loose this sharing when computing their closures.

Here we describe the second approach: this enables some additional optimisations, and the above drawback disappears as one can always take an overlapping union. (Note however that if the automata were actually disjoint, taking the closure after the union potentially results in bigger automata: the history part—$\mathcal{G}(Q)$—being shared, parts of one of the underlying automata can be duplicated just because of the history related to the other automaton.) We thus assume a non-deterministic automaton $(Q, \mathbf{X}, \mathscr{T}, \Delta)$ with two sets of initial states $I_\mathbf{e}$, $I_\mathbf{f}$ such that $[\![\mathbf{e}]\!] = L((Q, \mathbf{X}, I_\mathbf{e}, \mathscr{T}, \Delta))$ and $[\![\mathbf{f}]\!] = L((Q, \mathbf{X}, I_\mathbf{f}, \mathscr{T}, \Delta))$.

We thus need an algorithm for checking whether $(I_\mathbf{e}, \mathrm{Id}_Q)$ and $(I_\mathbf{f}, \mathrm{Id}_Q)$ are equivalent in the closure of $(Q, \mathbf{X}, \mathscr{T}, \Delta)$. Due to the shape of the determinisation of this automaton, we consider *stratified relations*, i.e., relations indexed by histories.

Definition 27. A *stratified relation* $\mathcal{R}$ is a function from histories in $\mathcal{G}(Q)$ to relations on sets of states:

$$\mathcal{R} : \mathcal{G}(Q) \to \mathcal{P}(\mathcal{P}(Q) \times \mathcal{P}(Q))$$

We write $P\, \mathcal{R}_R\, P'$ for $(P, P') \in \mathcal{R}(R)$.

One can then define an appropriate notion of (stratified) bisimulation:

Definition 28 *(Progression, bisimulation).* Given two stratified relations $\mathcal{R}, \mathcal{R}'$, we say that $\mathcal{R}$ *progresses to* $\mathcal{R}'$, denoted $\mathcal{R} \rightarrowtail \mathcal{R}'$, if whenever $P\, \mathcal{R}_R\, P'$ then

1. $P \cap \mathscr{T} = \emptyset$ if and only if $P' \cap \mathscr{T} = \emptyset$ and
2. for all $x \in \mathbf{X}$, $P \cdot \Delta(x) \cdot h_x(R)\, \mathcal{R}'_{h_x(R)}\, P' \cdot \Delta(x) \cdot h_x(R)$.

A *bisimulation* is a stratified relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{R}$.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

```
   input : Two regular expressions with converse e, f ∈ Reg_X^∨
   output: A Boolean, saying whether or not KAC ⊢ e = f.
1  ⟨Q, X, l_e, l_f, 𝒯, Δ⟩ ← Antimirov' automaton recognising [[e]] and [[f]];
2  L ← {(Id_Q, l_e, l_f)} ;                                   /* Set of elements to be processed */
3  ℛ ← (_ ↦ ∅) ;                                  /* Stratified relation meant to become a bisimulation */
4  while L ≠ ∅ do
5  |  // ℛ ↣ g(ℛ) ∪ L
6  |  Pick (R, P, P') from L;
7  |  if (P, P') ∉ g(ℛ)(R) then
8  |  |  if is_empty(P ∩ 𝒯) = is_empty(P' ∩ 𝒯) then
9  |  |  |  foreach x ∈ X do Add (h_x(R), P · Δ(x) · h_x(R), P' · Δ(x) · h_x(R)) to L
10 |  |  |  Add (P, P') to ℛ(R);
11 |  |  else
12 |  |  |  return false ;                                  /* A difference appeared for some word, e ≠ f */
13 |  |
14 |
15 end
16 return true ;                          /* There was no difference, ℛ is a bisimulation up to g, KAC ⊢ e = f */
```

Algorithm 2: Time-efficient algorithms for KAC, g may range over various up-to techniques.

Proposition 29 *(Coinduction). The languages $cl([\![\mathbf{e}]\!])$ and $cl([\![\mathbf{f}]\!])$ are equivalent if and only if there exists a bisimulation $\mathcal{R}$ such that $l_\mathbf{e}\ \mathcal{R}_{\mathrm{Id}_Q}\ l_\mathbf{f}$.*

Proof. Simple adaptation of the same result in [14], to work with stratified relations. □

Accordingly, we obtain Algorithm 2 (where we assume g to be the identity function, for now). This algorithm works as follows: the variable $\mathcal{R}$ contains a relation which is a bisimulation candidate and the variable L contains a queue of triples (R, P, Q) that remain to be processed (R being a history, and P, Q being sets of states). To process such a pair, one first checks whether it already belongs to the bisimulation candidate: in that case, the pair can be skipped since it was already processed. Otherwise, one checks that both sets are either accepting or non-accepting (line 8), and one adds all derivatives of the pair to L (line 9). The triple (R, P, Q) is finally added to the bisimulation candidate (line 10), and we proceed with the remainder of the queue. When the queue L becomes empty, then $\mathcal{R}$ is a bisimulation thanks to the main loop invariant (line 5—recall that for now, g is the identity function), so that the starting expression are equivalent.

This algorithm can be enhanced by exploiting *up-to techniques* [26,15]: an up-to technique is a function g on (stratified) relations such that any relation $\mathcal{R}$ satisfying $\mathcal{R} \rightarrowtail g(\mathcal{R})$ is contained in a bisimulation. Intuitively, such relations, that are not necessarily bisimulations, are constrained enough to contain only language equivalent pairs.

Examples of such up-to techniques include:

1. *equivalence closure* (in the present context of stratified relations, the function e associating to a stratified relation $\mathcal{R}$ the stratified relation $e(\mathcal{R})$ mapping any history R to the smallest equivalence relation that contains $\mathcal{R}(R)$). This technique is implicitly used in the algorithm by Hopcroft and Karp [21], via a disjoint-set forest data structure: we get their algorithm by choosing $g = e$ in Algorithm 2.
2. *congruence closure*: the function c mapping any history R to the smallest equivalence relation S that contains $\mathcal{R}(R)$ and that satisfies the following rule.

$$\frac{P_i\ S\ P_i',\ \ i = 1, 2}{(P_1 \cup P_2)\ S\ (P_1' \cup P_2')}$$

   This is the key up-to technique introduced in [14]; it allows one to avoid exploring large parts of the automaton, and to stop much earlier than with other algorithms (some sets of states that are accessible through the power-set construction need not be visited at all).

Proposition 30. *The above functions e and c are valid up-to techniques in the present setting.*

Proof. Following the same arguments as in [14]. In particular, it holds that $\mathcal{R} \rightarrowtail e(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail c(\mathcal{R})$) entails $e(\mathcal{R}) \rightarrowtail e(\mathcal{R})$ (resp. $c(\mathcal{R}) \rightarrowtail c(\mathcal{R})$). □

The resulting algorithms (Algorithm 2 with g instantiated with either e or c) require exponential time (and space) in worst case, as the final bisimulation candidate, $\mathcal{R}$, can be exponentially large. However in practice, like in the simpler case of Kleene algebra without converse, those worst cases are hard to reach, so that such algorithms can usually cope with expressions with up to a thousand of nodes [14].

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

6. Conclusion

Building on the work of Bernátsky, Bloom, Ésik and Stefanescu, we gave new and more efficient algorithms to decide the theory KAC. These algorithms rely on a more compact construction for the closure of an automaton. The first one (Algorithm 1) is PSPACE, which allowed us to show that the equational theory of KAC is PSPACE-complete. The other ones (the two main instances of Algorithm 2) are not PSPACE, but they work well in practice; they are variants of the standard Hopcroft and Karp's algorithm [21], and of its recent optimisation using bisimulations up to congruence [14].

To prove the correctness of the main automata construction, we used the family of regular languages $\Gamma(w)$ (corresponding to $G(w^\vee)$ in [4]). We established the main properties of this family using a proper finite automata characterisation. Moreover, this family allowed us to reformulate the proof of the completeness of the reduction from equality in $\mathrm{Rel}^\vee$ to equivalence of closed automata (implication (6) from the introduction).

As an exercise, we have implemented and tested the various constructions and algorithms in an OCAML program which is available online [17].

To continue this work, we would like to implement one of the presented algorithms in the proof assistant COQ, as a tactic to automatically prove the equalities in KAC—as it has already been done for the theories KA [27] and KAT [28]. The simplifications we propose in this paper give us hope that such a task is feasible. The main difficulty certainly lies in the formalisation of the completeness proof of KAC (implication (7) from the introduction), whose key step consists in proving that for all expression $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$, there exists a proof of $\mathbf{e} = cl(\mathbf{e})$ in KAC (where $cl(\mathbf{e})$ is a regular expression for the regular language $cl(\llbracket \mathbf{e} \rrbracket)$). This is established in [5], but the proof uses yet another automaton construction for the closure, which is even more complicated than the one used in [4], and which seems quite difficult to formalise in COQ. We hope to find an alternative completeness proof, by exploiting our simpler construction.

Acknowledgements

We are grateful to the anonymous referees of RAMiCS'14, who helped us to improve this paper and who suggested us the alternative proof of correctness which we provide in Section 5.1.

Appendix A. Proof of Equation (9)

We will show here that $\widehat{\eta}(e) = \widehat{\eta}(f)$ implies that $\llbracket e \rrbracket = \llbracket f \rrbracket$, for e and f regular expressions over $\mathbf{X}$.
It is well known that for any expression $e \in \mathrm{Reg}_{\mathbf{X}}$, for any $\sigma : \mathbf{X} \longrightarrow \mathcal{P}(\Sigma^\star)$,

$$\widehat{\sigma}(e) = \bigcup_{w \in \llbracket e \rrbracket} \widehat{\sigma}(w).$$

Consider the following partial function:

$$\begin{aligned} i : X^\star_\bullet &\longrightarrow \mathbf{X}^\star \\ \epsilon &\longmapsto \epsilon \\ x\bullet w &\longmapsto x \cdot i(w) \\ \bullet x w &\longmapsto x' \cdot i(w)\ . \end{aligned}$$

We will write $\widehat{i}$ for the function $[W \mapsto \{i(w) \mid w \in W\}]$. We will show by induction on $w \in \mathbf{X}^\star$ that $\widehat{i} \cdot \widehat{\eta}(w) = \{w\}$:

- $\widehat{i} \cdot \widehat{\eta}(\epsilon) = \widehat{i}(\{\epsilon\}) = \{\epsilon\}$;
- if $x \in X$, then

$$\begin{aligned} \widehat{i} \cdot \widehat{\eta}(xw) &= \widehat{i}(\eta(x) \cdot \widehat{\eta}(w)) && (\widehat{\eta} \text{ is a morphism}) \\ &= \widehat{i}(\{x\bullet\} \cdot \widehat{\eta}(w)) && (\text{definition of } \eta) \\ &= \{x\} \cdot (\widehat{i} \cdot \widehat{\eta}(w)) && (\text{definition of } i) \\ &= \{xw\}; && (\text{induction hypothesis}) \end{aligned}$$

- and similarly if $x' \in X'$, then $\widehat{i} \cdot \widehat{\eta}(x'w) = \widehat{i}(\{\bullet x\} \cdot \widehat{\eta}(w)) = \{x'\} \cdot (\widehat{i} \cdot \widehat{\eta}(w)) = \{x'w\}$.

Thus, we get that:

$$\llbracket e \rrbracket = \bigcup_{w \in \llbracket e \rrbracket} \{w\} = \bigcup_{w \in \llbracket e \rrbracket} \widehat{i} \cdot \widehat{\eta}(w) = \widehat{i}\left(\bigcup_{w \in \llbracket e \rrbracket} \widehat{\eta}(w) \right) = \widehat{i}(\widehat{\eta}(e)).$$

Thus we get $\llbracket e \rrbracket = \widehat{i}(\widehat{\eta}(e)) = \widehat{i}(\widehat{\eta}(f)) = \llbracket f \rrbracket$.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

Appendix B. Proof of Proposition (8)

Let us prove the first implication of Proposition 8:

$$\forall \mathbf{w} \in \mathbf{X}^\star, \forall u \in \Gamma(\mathbf{w}), \exists v \in \text{suffixes}(\mathbf{w}) : u \rightsquigarrow^\star \overline{v} v.$$

We will proceed by induction on $\mathbf{w}$:

1. If $\mathbf{w} = \epsilon$, then $u \in \Gamma(\epsilon) = \{\epsilon\}$. So $u = \epsilon \rightsquigarrow^0 \overline{\epsilon}\epsilon$ and obviously $\epsilon \in \text{suffixes}(\epsilon)$.
2. Otherwise $\mathbf{w} = wx$, and $u \in \Gamma(wx) = \left(x'\Gamma(w)x\right)^\star$. Thus we know that for some $n \in \mathbb{N}$, $u \in \left(x'\Gamma(w)x\right)^n$. We now will prove by recurrence on n that $u \in \left(x'\Gamma(w)x\right)^n \Rightarrow \exists v \in \text{suffixes}(wx) : u \rightsquigarrow^\star \overline{v} v$:
   (a) If $n = 0$ then $u = \epsilon \rightsquigarrow^0 \overline{\epsilon}\epsilon$ and $\epsilon \in \text{suffixes}(wx)$.
   (b) If $n = m + 1$ then we can introduce $u_1 \in \Gamma(w)$ and $u_2 \in (x'\Gamma(w)x)^m$ such that $u = x'u_1xu_2$.
      i. By induction hypothesis, $\exists v_1 \in \text{suffixes}(w)$ such that $u_1 \rightsquigarrow^\star \overline{v_1} v_1$.
      ii. By recurrence hypothesis, $\exists v_2 \in \text{suffixes}(wx)$ such that $u_2 \rightsquigarrow^\star \overline{v_2} v_2$.

   Thus we know that $u = x'u_1xu_2 \rightsquigarrow^\star x'\overline{v_1}v_1x\overline{v_2}v_2$. We will now do a case analysis on the length of v_2.
      i. If $|v_2| = 0$, then $v_2 = \epsilon$ so $u \rightsquigarrow^\star x'\overline{v_1}v_1x = \overline{v_1x}v_1x$.
      ii. If $|v_2| > 0$, as $v_2 \in \text{suffixes}(wx)$, we can write $v_2 = v_3x$ with $v_3 \in \text{suffixes}(w)$. We will now compare the sizes of v_1 and v_3, both being suffixes of w.
         A. If $|v_1| \leqslant |v_3|$, then $v_3 = v_4v_1$. Thus we have:

$$\begin{aligned} u &\rightsquigarrow^\star x'\overline{v_1}v_1x\overline{v_3x}v_3x = x'\overline{v_1}v_1xx'\overline{v_1}\ \overline{v_4}v_4v_1x \\ &= \overline{v_1x}v_1x\overline{v_1x}\ \overline{v_4}v_4v_1x \\ &\rightsquigarrow \overline{v_1x}\ \overline{v_4}v_4v_1x = \overline{v_2}v_2 \end{aligned}$$

         B. Otherwise we can write $v_1 = v_5v_3$ and thus:

$$\begin{aligned} u &\rightsquigarrow^\star x'\overline{v_5v_3}v_5v_3x\overline{v_3x}v_3x \\ &\rightsquigarrow \overline{v_3v_5x}v_5v_3x = \overline{v_1x}v_1x \end{aligned}$$

   So we have shown that either $u \rightsquigarrow^\star \overline{v_1x}v_1x$ or $u \rightsquigarrow^\star \overline{v_2}v_2$, and as we know that both v_1x and v_2 are suffixes of wx, we have finished.

Appendix C. Bisimulation relating the two constructions

Let us be more precise: starting from a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, Q_f, \Delta\rangle$, its determinised is $\mathcal{D} = \langle \mathcal{P}(Q), \mathbf{X}, I, \mathscr{T}, \delta\rangle$ with

$$\mathscr{T} = \left\{P \;\middle|\; P \cap Q_f \neq \emptyset\right\} \text{ and } \delta(P, x) = P \cdot \Delta(x).$$

We can build two automata recognising its closure. The first one, derived from our construction, is

$$\mathscr{A}_1 = \langle \mathcal{P}(Q) \times \mathcal{G}(Q), \mathbf{X}, (I, \mathrm{Id}_Q), \mathscr{T}_1, \delta_1\rangle$$

where $\mathcal{G}(Q)$ is the set reflexive transitive relations over Q,

$$\mathscr{T}_1 \triangleq \{(P, R) \mid P \cap Q_f \neq \emptyset,\ R \in \mathcal{G}(Q)\},$$

and $\delta_1((P, R), x) = (P \cdot h_x(R), h_x(R))$.

The second one, given by the original construction, is

$$\mathscr{A}_2 = \langle \mathcal{P}(M_{\mathcal{D}}) \times \mathcal{P}(M_{\mathcal{D}}), \mathbf{X}, (\underline{\epsilon}, \underline{\epsilon}), \mathscr{T}_2, \delta_2\rangle$$

where $M_{\mathcal{D}}$ is the transition monoid of $\mathcal{D}$, a set of endomorphisms of $\mathcal{P}(Q)$ induced by words, $\underline{w} \triangleq \{w_{\mathcal{D}}\}$ is a singleton containing the interpretation of a word w in $M_{\mathcal{D}}$, $\mathscr{T}_2 \triangleq \{(F, G) \mid \exists q_f \in Q_f, \exists f \in F : q_f \in f(I)\}$, and the transition function is

$$\delta_2((F, G), x) = (F \odot \underline{x} \odot (\underline{x}' \odot G \odot \underline{x})^\star, (\underline{x}' \odot G \odot \underline{x})^\star).$$

($A \odot B \triangleq \{g \cdot f \mid f \in A \wedge g \in B\}$.) The fact that the elements of $M_{\mathcal{D}}$ are semilattice-homomorphisms can be easily checked, as $u_{\mathcal{D}}(P)$ is the only state of $\mathcal{D}$ (*i.e.* a set of states of $\mathscr{A}$) such that $P \xrightarrow{u}_{\mathcal{D}} u_{\mathcal{D}}(P)$. Then is straightforward that:

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

$$\begin{aligned} u_{\mathcal{D}}(P_1 \cup P_2) &= \{q \in Q \mid \exists p \in P_1 \cup P_2 : p \xrightarrow{u}_{\mathscr{A}} q\} \\ &= \{q \in Q \mid \exists p \in P_1 : p \xrightarrow{u}_{\mathscr{A}} q\} \cup \{q \in Q \mid \exists p \in P_2 : p \xrightarrow{u}_{\mathscr{A}} q\} \\ &= u_{\mathcal{D}}(P_1) \cup u_{\mathcal{D}}(P_2). \end{aligned}$$

Now, to give the bisimulation we need the following morphism i from $\mathcal{P}(M_{\mathcal{D}})$ to $\mathcal{P}\left(Q^2\right)$ defined by

$$i(F) \triangleq \{(p,q) \mid \exists f \in F : q \in f(\{p\})\}.$$

Note that i is a KA-homomorphism because the elements of the transition monoid of the determinised automaton are semilattice-homomorphisms from $\mathcal{P}(Q)$ to $\mathcal{P}(Q)$. Let's check that:

$$\begin{aligned} \epsilon_{\mathcal{D}} &= \mathrm{Id}_{\mathcal{P}(Q)}, \text{ meaning that } i(\underline{\epsilon}) = \mathrm{Id}_Q; \\ i(F_1 \cup F_2) &= \{(p,q) \mid \exists f \in F_1 \cup F_2 : q \in f(\{p\})\} \\ &= \{(p,q) \mid \exists f \in F_1 : q \in f(\{p\})\} \cup \{(p,q) \mid \exists f \in F_2 : q \in f(\{p\})\} \\ &= i(F_1) \cup i(F_2); \\ i(F_1 \odot F_2) &= \{(p,q) \mid \exists f \in F_1 \odot F_2 : q \in f(\{p\})\} \\ &= \{(p,q) \mid \exists f, g \in F_1 \times F_2 : q \in g \cdot f(\{p\})\} \\ &= \{(p,q) \mid \exists f \in F_1 : \exists p' \in f(\{p\}) : \exists g \in F_2 : q \in g(\{p'\})\} && (g \text{ is a semilattice homomorphism}) \\ &= \{(p,q) \mid \exists p' : (p,p') \in i(F_1) \wedge (p',q) \in i(F_2)\} \\ &= i(F_1) \cdot i(F_2) \end{aligned}$$

For the * operation, recall that

$$\forall F \in \mathcal{P}(M_{\mathcal{D}}), \exists n_1(F) \in \mathbb{N} : \forall n_1(F) \leqslant m, F^\star = (F \cup \underline{\epsilon})^m;$$

and that

$$\forall R \in \mathcal{P}(Q)^2, \exists n_2(R) \in \mathbb{N} : \forall n_2(R) \leqslant m, R^\star = (R \cup \mathrm{Id}_Q)^m.$$

Then, if we write $m = \max(n_1(F), n_2(u_{\mathcal{D}}(F)))$,

$$\begin{aligned} i(F^\star) &= i((F \cup \underline{\epsilon})^m) \\ &= (i(F) \cup i(\underline{\epsilon}))^m \\ &= (i(F))^\star \end{aligned}$$

We can also check that, for any $x \in \mathbf{X}$:

$$\begin{aligned} i\left(\underline{x}\right) &= \{(p,q) \mid q \in x_{\mathcal{D}}(\{p\})\} \\ &= \{(p,q) \mid q \in \delta(\{p\}, x)\} \\ &= \{(p,q) \mid p \xrightarrow{x}_{\mathscr{A}} q\} \\ &= \Delta(x). \end{aligned}$$

The bisimulation $\sim$ can thus be expressed:

$$\sim \triangleq \{((I \cdot i(F), i(G)),\ (F,G))\}$$

where (F,G) are states of $\mathscr{A}_2$. We now prove that it is indeed a bisimulation.

1. We need the initial states to be related. This is obvious as $\epsilon_{\mathcal{D}} = \mathrm{Id}_{\mathcal{P}(Q)}$, meaning that $i(\underline{\epsilon}) = \mathrm{Id}_Q$. Furthermore, $[\epsilon] = \mathrm{Id}_Q$ and $I = I \cdot \mathrm{Id}_Q$. That means $(I, [\epsilon]) \sim (\underline{\epsilon}, \underline{\epsilon})$.
2. For the final states, it isn't much more complicated:

$$\begin{aligned} (F,G) \in \mathscr{T}_2 &\Leftrightarrow \exists q_f \in Q_f : \exists f \in F : q_f \in f(I) \\ &\Leftrightarrow \exists q_f \in Q_f : q_f \in I \cdot i(F) \\ &\Leftrightarrow I \cdot i(F) \cap Q_f \neq \emptyset \\ &\Leftrightarrow (I \cdot i(F), i(G)) \in \mathscr{T}_1. \end{aligned}$$

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005

3. What remains to be shown is that this relation is stable under transitions from both sides. Suppose that $(Q, R) \sim (F, G)$, and consider $x \in \mathbf{X}$. After reading x we get in $\mathscr{A}_2$ $(F \odot \underline{x} \odot G', G')$, with $G' = (\underline{x'} \odot G \odot \underline{x})^\star$, and in $\mathscr{A}_1$ $(Q \cdot (\Delta(x) \cdot h_x(R)), h_x(R))$. We will prove that they are still related in two steps, first by looking at the second component, and then dealing with the first one.
   (a) We know that $R = i(G)$, and that $i(\underline{x}) = \Delta(x)$.

$$\begin{aligned} h_x(R) &= (\Delta(x') \cdot R \cdot \Delta(x))^\star \\ &= (i(\underline{x'}) \cdot i(G) \cdot i(\underline{x}))^\star \\ &= i(G') \qquad\qquad (i \text{ is a morphism}) \end{aligned}$$

   (b) Now the first component comes quite easily:

$$\begin{aligned} Q \cdot (\Delta(x) \cdot h_x(R)) &= (I \cdot i(F)) \cdot (i(\underline{x}) \cdot i(G')) \\ &= I \cdot (i(F) \cdot i(\underline{x}) \cdot i(G')) \\ &= I \cdot i(F \odot \underline{x} \odot G'). \end{aligned}$$

References

[1] P. Brunet, D. Pous, Kleene algebra with converse, in: Proc. RAMiCS, in: Lecture Notes in Computer Science, vol. 8428, Springer-Verlag, 2014, pp. 101–118.
[2] J.H. Conway, Regular Algebra and Finite Machines, Chapman and Hall Mathematics Series, 1971.
[3] S.C. Kleene, Representation of events in nerve nets and finite automata, Memorandum, Rand Corporation, 1951.
[4] S.L. Bloom, Z. Ésik, G. Stefanescu, Notes on equational theories of relations, Algebra Univers. 33 (1) (1995) 98–126.
[5] Z. Ésik, L. Bernátsky, Equational properties of Kleene algebras of relations with conversion, Theor. Comput. Sci. 137 (2) (1995) 237–251.
[6] V.N. Redko, On defining relations for the algebra of regular events, Ukr. Mat. Zh. (1964) 120–126.
[7] D. Krob, A complete system of B-rational identities, in: Proc. ICALP, in: Lecture Notes in Computer Science, vol. 443, Springer-Verlag, 1990, pp. 60–73.
[8] A. Salomaa, Two complete axiom systems for the algebra of regular events, J. ACM 13 (1) (1966) 158–169.
[9] M. Boffa, Une remarque sur les systèmes complets d'identités rationnelles, Inform. Théor. Appl. 24 (1990) 419–428.
[10] D. Kozen, A completeness theorem for Kleene Algebras and the algebra of regular events, in: Proc. LICS, IEEE Computer Society, 1991, pp. 214–225.
[11] M. Boffa, Une condition impliquant toutes les identités rationnelles, Inform. Théor. Appl. 29 (6) (1995) 515–518.
[12] P. Freyd, A. Scedrov, Categories, Allegories, North-Holland, 1990.
[13] H. Andréka, D. Bredikhin, The equational theory of union-free algebras of relations, Algebra Univers. 33 (4) (1995) 516–532.
[14] F. Bonchi, D. Pous, Checking NFA equivalence with bisimulations up to congruence, in: Proc. POPL, ACM, 2013, pp. 457–468.
[15] D. Pous, D. Sangiorgi, Enhancements of the coinductive proof method, in: Advanced Topics in Bisimulation and Coinduction, Cambridge University Press, 2011, pp. 233–289.
[16] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1999.
[17] P. Brunet, D. Pous, Web appendix to this abstract, http://perso.ens-lyon.fr/paul.brunet/kac.html, 2014.
[18] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[19] A. Meyer, L.J. Stockmeyer, Word problems requiring exponential time, in: Proc. ACM Symposium on Theory of Computing, ACM, 1973, pp. 1–9.
[20] V.M. Glushkov, The abstract theory of automata, Russ. Math. Surv. 16 (5) (1961) 1.
[21] J.E. Hopcroft, R.M. Karp, A linear algorithm for testing equivalence of finite automata, Tech. rep., Cornell University, 1971.
[22] M.D. Wulf, L. Doyen, T.A. Henzinger, J.-F. Raskin, Antichains: a new algorithm for checking universality of finite automata, in: Proc. CAV, in: Lecture Notes in Computer Science, vol. 4144, Springer-Verlag, 2006, pp. 17–30.
[23] P.A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, T. Vojnar, When simulation meets antichains, in: Proc. TACAS, in: Lecture Notes in Computer Science, vol. 6015, Springer-Verlag, 2010, pp. 158–174.
[24] L. Doyen, J.-F. Raskin, Antichain algorithms for finite automata, in: Proc. TACAS, in: Lecture Notes in Computer Science, vol. 6015, Springer-Verlag, 2010, pp. 2–22.
[25] V.M. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, Theor. Comput. Sci. 155 (2) (1996) 291–319.
[26] D. Sangiorgi, On the bisimulation proof method, Math. Struct. Comput. Sci. 8 (1998) 447–479.
[27] T. Braibant, D. Pous, Deciding Kleene algebras in Coq, Log. Methods Comput. Sci. 8 (1) (2012) 1–16.
[28] D. Pous, Kleene algebra with tests and Coq tools for while programs, in: Proc. ITP, in: Lecture Notes in Computer Science, vol. 7998, Springer-Verlag, 2013, pp. 180–196.

Please cite this article in press as: P. Brunet, D. Pous, Algorithms for Kleene algebra with converse, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.07.005