Kleene Algebra with Converse

Talk at RAMICS '14

Paul Brunet & Damien Pous
LIP, CNRS, ENS de Lyon, INRIA, Université de Lyon, UMR 5668

April 28th, 2014

Introduction

$$(x^\star + y) \cdot z \qquad\qquad (x^\star \cdot z) + (y \cdot z)$$

Introduction

$$\forall S, \forall \sigma : \mathcal{R}eg_X \to \mathcal{R}el(S)$$

$$\begin{tikzcd}
(x^\star + y) \cdot z \arrow[d] & & (x^\star \cdot z) + (y \cdot z) \arrow[d] \\
\sigma\left((x^\star + y) \cdot z\right) & = & \sigma\left((x^\star \cdot z) + (y \cdot z)\right)
\end{tikzcd}$$

Introduction

$$\forall S, \forall \sigma : \mathcal{R}eg_X \to \mathcal{R}el(S)$$

$$\begin{tikzcd}
(x^\star + y) \cdot z \arrow[d] & \equiv_{\mathcal{R}el} & (x^\star \cdot z) + (y \cdot z) \arrow[d] \\
\sigma\left((x^\star + y) \cdot z\right) & = & \sigma\left((x^\star \cdot z) + (y \cdot z)\right)
\end{tikzcd}$$

Introduction

$$\begin{tikzcd}
\mathrm{KA} \vdash e = f \arrow[rr, Leftrightarrow, "{e, f \in \mathcal{R}eg_X}"] \arrow[dr, Leftrightarrow] & & e \equiv_{\mathcal{R}el} f \arrow[dl, Leftrightarrow] \\
& \llbracket e \rrbracket = \llbracket f \rrbracket &
\end{tikzcd}$$

Introduction

$$\begin{tikzcd}
\mathrm{KA} \vdash e = f \arrow[rr, Leftrightarrow, "{e, f \in \mathcal{R}eg_X}"] \arrow[dr, Leftrightarrow] & & e \equiv_{\mathcal{R}el} f \\
& \llbracket e \rrbracket = \llbracket f \rrbracket \arrow[ur, Leftrightarrow] &
\end{tikzcd}$$

What if we add a *converse* operation to regular expressions ?

Introduction

$$\begin{tikzcd}
\mathrm{KA} \vdash e = f \arrow[rr, Leftrightarrow, "{e, f \in \mathcal{R}eg_X}"] \arrow[dr, Leftrightarrow] & & e \equiv_{\mathcal{R}el} f \arrow[dl, Leftrightarrow] \\
& [\![e]\!] = [\![f]\!] &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{KAC} \vdash e = f \arrow[rr, Leftrightarrow, "{e, f \in \mathcal{R}eg_X{}^{\vee}}"] \arrow[dr, Leftrightarrow] & & e \equiv_{\mathcal{R}el^{\vee}} f \arrow[dl, Leftrightarrow] \\
& \mathcal{cl}([\![\mathbf{e}]\!]) = \mathcal{cl}([\![\mathbf{f}]\!]) &
\end{tikzcd}$$

Introduction

$$e, f \in \mathcal{R}eg_X{}^{\vee}$$

$$\begin{tikzcd}
 & e \equiv_{\mathcal{R}el^{\vee}} f \\
cl\left(\llbracket \mathbf{e} \rrbracket\right) = cl\left(\llbracket \mathbf{f} \rrbracket\right) \arrow[ur, Leftrightarrow] &
\end{tikzcd}$$

Plan

1. Introduction
2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra with converse
   - Reduction to an automaton problem
3. Closure of an automaton
4. The PSPACE algorithm.
5. Conclusion

Plan

1. Introduction
2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra with converse
   - Reduction to an automaton problem
3. Closure of an automaton
4. The PSPACE algorithm.
5. Conclusion

Regular expressions with converse

Regular expressions with converse over X

$$e, f \in \mathcal{R}eg_X{}^{\vee} ::= \mathbb{0} \mid \mathbb{1} \mid x \in X \mid e + f \mid e \cdot f \mid e^{\star} \mid e^{\vee}$$

Regular expressions with converse

Regular expressions with converse over X

$$e, f \in \mathcal{R}eg_X{}^{\vee} ::= \mathbb{0} \mid \mathbb{1} \mid x \in X \mid e + f \mid e \cdot f \mid e^{\star} \mid e^{\vee}$$

Given any map :

$$\sigma : X \longrightarrow \mathcal{R}el(S)\,,$$

we can build uniquely a morphism

$$\hat{\sigma} : \mathcal{R}eg_X{}^{\vee} \longrightarrow \mathcal{R}el(S)\,.$$

Relational equivalence

For $e, f \in \mathcal{R}eg_X{}^{\vee}$:

$$e \equiv_{\mathcal{R}el^{\vee}} f$$

means that

$$\forall S, \forall \sigma : X \to \mathcal{R}el(S), \hat{\sigma}(e) = \hat{\sigma}(f).$$

The equational theory KAC

The equational theory KAC of regular algebras with converse over binary relations consists of the axioms of KA together with the following :

$$(a+b)^{\vee} = a^{\vee} + b^{\vee} \tag{1}$$

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

$$(a^{\star})^{\vee} = (a^{\vee})^{\star} \tag{3}$$

$$a^{\vee\vee} = a \tag{4}$$

$$aa^{\vee}a \geqslant a \tag{5}$$

The equational theory KAC

The equational theory KAC of regular algebras with converse over binary relations consists of the axioms of KA together with the following :

$$(a+b)^\vee = a^\vee + b^\vee \tag{1}$$

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

$$(a^\star)^\vee = (a^\vee)^\star \tag{3}$$

$$a^{\vee\vee} = a \tag{4}$$

$$aa^\vee a \geqslant a \tag{5}$$

From $\mathcal{R}eg_X{}^\vee$ to $\mathcal{R}eg_{\mathbf{X}}$

Let X be a finite alphabet. For $e \in \mathcal{R}eg_X$, we write $[\![e]\!] \subseteq X^\star$ for the *language denoted by* e.

- $X' := \{x' \mid x \in X\}$ is a disjoint copy of X,
- and $\mathbf{X} := X \cup X'$.

We apply the following rewriting system :

$$\begin{cases} (a+b)^\vee \longmapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \longmapsto b^\vee \cdot a^\vee \\ (a^\star)^\vee \longmapsto (a^\vee)^\star \\ a^{\vee\vee} \longmapsto a \end{cases}$$

From $\mathcal{R}eg_X{}^\vee$ to $\mathcal{R}eg_{\mathbf{X}}$

Let X be a finite alphabet. For $e \in \mathcal{R}eg_X$, we write $[\![e]\!] \subseteq X^\star$ for the *language denoted by* e.

- $X' := \{x' \mid x \in X\}$ is a disjoint copy of X,
- and $\mathbf{X} := X \cup X'$.

We apply the following rewriting system :

$$\begin{cases} (a+b)^\vee \longmapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \longmapsto b^\vee \cdot a^\vee \\ (a^\star)^\vee \longmapsto (a^\vee)^\star \\ a^{\vee\vee} \longmapsto a \\ x^\vee \longmapsto x' \quad (x \in X) \\ x'^\vee \longmapsto x \quad (x \in X) \end{cases}$$

From $\mathcal{R}eg_X{}^{\vee}$ to $\mathcal{R}eg_{\mathbf{X}}$

Let X be a finite alphabet. For $e \in \mathcal{R}eg_X$, we write $[\![e]\!] \subseteq X^\star$ for the *language denoted by e*.

- $X' := \{x' \mid x \in X\}$ is a disjoint copy of X,
- and $\mathbf{X} := X \cup X'$.

We apply the following rewriting system :

$$\begin{cases} (a+b)^{\vee} \longmapsto a^{\vee} + b^{\vee} \\ (a \cdot b)^{\vee} \longmapsto b^{\vee} \cdot a^{\vee} \\ (a^\star)^{\vee} \longmapsto (a^{\vee})^\star \\ a^{\vee\vee} \longmapsto a \\ x^{\vee} \longmapsto x' \quad (x \in X) \\ x'^{\vee} \longmapsto x \quad (x \in X) \\ \mathbb{1}^{\vee} \longmapsto \mathbb{1} \\ \mathbb{0}^{\vee} \longmapsto \mathbb{0} \end{cases}$$

From $\mathcal{R}eg_X{}^\vee$ to $\mathcal{R}eg_{\mathbf{X}}$

Let X be a finite alphabet. For $e \in \mathcal{R}eg_X$, we write $[\![e]\!] \subseteq X^\star$ for the *language denoted by* e.

- $X' := \{x' \mid x \in X\}$ is a disjoint copy of X,
- and $\mathbf{X} := X \cup X'$.

We apply the following rewriting system :

$$\begin{cases} (a+b)^\vee \longmapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \longmapsto b^\vee \cdot a^\vee \\ (a^\star)^\vee \longmapsto (a^\vee)^\star \\ a^{\vee\vee} \longmapsto a \\ x^\vee \longmapsto x' \quad (x \in X) \\ x'^\vee \longmapsto x \quad (x \in X) \\ \mathbb{1}^\vee \longmapsto \mathbb{1} \\ \mathbb{0}^\vee \longmapsto \mathbb{0} \end{cases}$$

We get $\mathbf{e} \in \mathcal{R}eg_{\mathbf{X}}$.

Is it enough ?

$$e, f \in \mathcal{R}eg_X \quad : \qquad e \equiv_{\mathcal{R}el} f \qquad \begin{matrix} \Rightarrow \\ \Leftarrow \end{matrix} \qquad [\![e]\!] = [\![f]\!]$$

Is it enough ?

$$e, f \in \mathcal{R}eg_X{}^{\vee} : \qquad e \equiv_{\mathcal{R}el^{\vee}} f \quad \Leftarrow \quad [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$$

Is it enough ?

$$e, f \in \mathcal{R}eg_X{}^{\vee} : \qquad e \equiv_{\mathcal{R}el^{\vee}} f \qquad \begin{array}{c}\not\Rightarrow\\ \Leftarrow\end{array} \qquad [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$$

$$aa^{\vee}a \geqslant a$$

$$\begin{tikzcd} x \arrow[r, bend left, "R"] & y \arrow[l, bend left, "R^{\vee}"] \end{tikzcd}$$

$e = aa^{\vee}a, f = a$:

$$[\![\mathbf{e}]\!] = \{aa'a\} \not\supseteq \{a\} = [\![\mathbf{f}]\!]$$

Is it enough ?

$$e, f \in \mathcal{R}eg_X{}^{\vee} : \qquad e \equiv_{\mathcal{R}el^{\vee}} f \qquad \begin{matrix}\Rightarrow \\ \Leftarrow\end{matrix} \qquad cl(\llbracket \mathbf{e} \rrbracket) = cl(\llbracket \mathbf{f} \rrbracket)$$

$$aa^{\vee}a \geqslant a$$

$$\begin{tikzcd} x \arrow[r, bend left, "R"] & y \arrow[l, bend left, "R^{\vee}"] \end{tikzcd}$$

$e = aa^{\vee}a, f = a$:

$$\llbracket \mathbf{e} \rrbracket = \{aa'a\} \not\supseteq \{a\} = \llbracket \mathbf{f} \rrbracket$$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$:

$$\begin{array}{ll|l} \forall x \in X, & \overline{x} := x' & \overline{\epsilon} := \epsilon \\ \forall x' \in X', & \overline{x'} := x & \overline{wx} := \overline{x}\ \overline{w}. \end{array}$$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$:

$$\begin{array}{rl} \forall x \in X, & \overline{x} := x' \\ \forall x' \in X', & \overline{x'} := x \end{array} \left| \begin{array}{rl} \overline{\epsilon} & := \epsilon \\ \overline{wx} & := \overline{x}\ \overline{w}. \end{array}\right.$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$:

$$\begin{array}{rl|l} \forall x \in X, & \overline{x} := x' & \overline{\epsilon} := \epsilon \\ \forall x' \in X', & \overline{x'} := x & \overline{wx} := \overline{x}\ \overline{w}. \end{array}$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

$$cl(L) := \{v \in \mathbf{X}^\star \mid \exists u \in L : u \rightsquigarrow^\star v\}$$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$: $\begin{array}{ll} \forall x \in X, & \overline{x} := x' \\ \forall x' \in X', & \overline{x'} := x \end{array} \left| \begin{array}{l} \overline{\epsilon} := \epsilon \\ \overline{wx} := \overline{x}\ \overline{w}. \end{array} \right.$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

$$cl(L) := \{v \in \mathbf{X}^\star \mid \exists u \in L : u \rightsquigarrow^\star v\}$$

Example :

$aa'a = a\bar{a}a \rightsquigarrow a$, so we have :$cl(\{aa'a\}) = \{a, aa'a\} \supseteq \{a\}$.

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$: $\begin{array}{ll} \forall x \in X, & \overline{x} := x' \\ \forall x' \in X', & \overline{x'} := x \end{array} \Bigg| \begin{array}{l} \overline{\epsilon} := \epsilon \\ \overline{wx} := \overline{x}\ \overline{w}. \end{array}$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Closure of a language : $\mathcal{cl}(L)$

$$\mathcal{cl}(L) := \{v \in \mathbf{X}^\star \mid \exists u \in L : u \rightsquigarrow^\star v\}$$

Example :

$abbabb'a'abbaa' = abb \cdot ab \cdot b'a' \cdot ab \cdot baa'$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$: $\begin{array}{ll} \forall x \in X, & \overline{x} := x' \\ \forall x' \in X', & \overline{x'} := x \end{array} \Bigg| \begin{array}{l} \overline{\epsilon} := \epsilon \\ \overline{wx} := \overline{x}\ \overline{w}. \end{array}$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

$$cl(L) := \{v \in \mathbf{X}^\star \mid \exists u \in L : u \rightsquigarrow^\star v\}$$

Example :

$abbabb'a'abbaa' = abb \cdot ab \cdot b'a' \cdot ab \cdot baa' = abb \cdot ab \cdot \overline{ab} \cdot ab \cdot baa'$

Reduction relation and closure

Converse for words : $\overline{w}$

For a word $w \in \mathbf{X}^\star$, we define inductively $\overline{w}$: $\begin{array}{ll} \forall x \in X, & \overline{x} := x' \\ \forall x' \in X', & \overline{x'} := x \end{array} \Bigg| \begin{array}{l} \overline{\epsilon} := \epsilon \\ \overline{wx} := \overline{x}\ \overline{w}. \end{array}$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w\overline{w}w \cdot u_2 \quad \rightsquigarrow \quad u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

$$cl(L) := \{v \in \mathbf{X}^\star \mid \exists u \in L : u \rightsquigarrow^\star v\}$$

Example :

$abbabb'a'abbaa' = abb \cdot ab \cdot b'a' \cdot ab \cdot baa' = abb \cdot ab \cdot \overline{ab} \cdot ab \cdot baa' \rightsquigarrow abb \cdot ab \cdot baa'$

Closure

Theorem [a]

$$e \equiv_{\mathcal{R}el^{\vee}} f \quad \Leftrightarrow \quad cl(\llbracket \mathbf{e} \rrbracket) = cl(\llbracket \mathbf{f} \rrbracket)$$

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

Plan

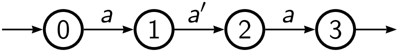
1. Introduction
2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra with converse
   - Reduction to an automaton problem
3. Closure of an automaton
4. The PSPACE algorithm.
5. Conclusion

Problem

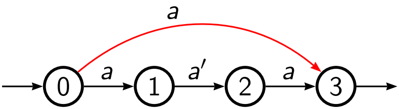
Input : an automaton $\mathscr{A}$ over $\mathbf{X}$

Output : an automaton $\mathscr{A}'$ over $\mathbf{X}$ such that $L(\mathscr{A}') = \mathscr{cl}(L(\mathscr{A}))$.

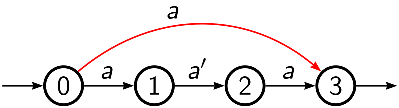
Intuition

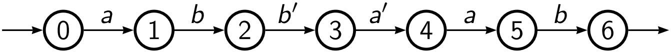


Intuition

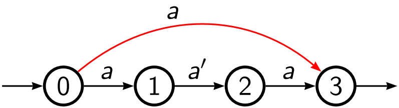


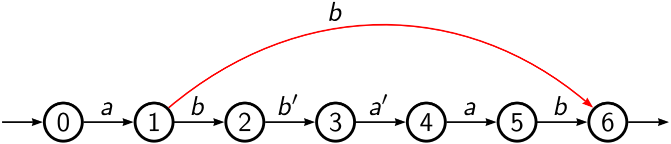
Intuition



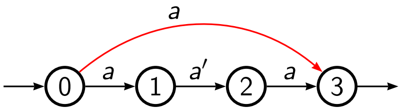


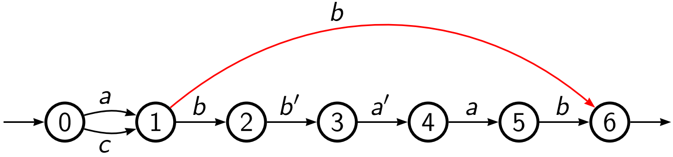
Intuition



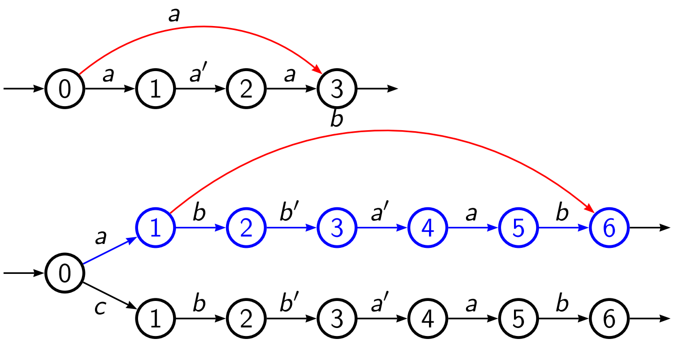


Intuition





Intuition



General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).

General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).
- We give an alternative construction, much lighter. The states of our automaton will be pairs of
  - a state of the initial automaton
  - and some history.

General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).
- We give an alternative construction, much lighter. The states of our automaton will be pairs of
  - a state of the initial automaton (q)
  - and some history (H).
- We will do a transition $(q_1, H) \xrightarrow{a} (q_2, H')$ if :

General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).
- We give an alternative construction, much lighter. The states of our automaton will be pairs of
  - a state of the initial automaton (q)
  - and some history (H).
- We will do a transition $(q_1, H) \xrightarrow{a} (q_2, H')$ if :
  - $q_1 \xrightarrow{a} q_3$,

General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).
- We give an alternative construction, much lighter. The states of our automaton will be pairs of
  - a state of the initial automaton (q)
  - and some history (H).
- We will do a transition $(q_1, H) \xrightarrow{a} (q_2, H')$ if :
  - $q_1 \xrightarrow{a} q_3$,
  - $H \xmapsto{a} H'$,

General idea

- Bloom, Ésik and Stefanescu give a construction, using the transitions monoid of the initial automaton, building a deterministic automaton with size $2^{2^{n^2}}$ (if the initial automaton has size n).
- We give an alternative construction, much lighter. The states of our automaton will be pairs of
  - a state of the initial automaton (q)
  - and some history (H).
- We will do a transition $(q_1, H) \xrightarrow{a} (q_2, H')$ if :
  - $q_1 \xrightarrow{a} q_3$,
  - $H \xmapsto{a} H'$,
  - and H' allows to jump from q_3 to q_2.

$\Gamma(w)$

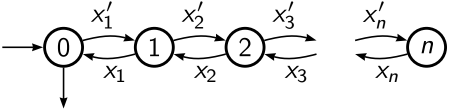
Definition : $\Gamma(w)$

$$\Gamma(\epsilon) = \{\epsilon\}$$
$$\Gamma(wx) = (\{x'\} \cdot \Gamma(w) \cdot \{x\})^\star$$

Lemma

$$u \in \Gamma(w) \Leftrightarrow \exists v \in \text{suffixes}(w) : u \rightsquigarrow^\star \overline{v}v$$

$\Gamma(x_n \cdots x_1)$ is recognised by the automaton :



$\gamma(w)$

Consider an automaton $\mathscr{A} = \langle Q, A, I, T, \Delta \rangle$, we write $\Delta_x := \{(p, q) \mid p \xrightarrow{x} q \in \Delta\}$.

Definition : $\gamma(w)$

$$\gamma(\epsilon) = \mathsf{Id}_Q$$
$$\gamma(wx) = (\Delta_{x'} \circ \gamma(w) \circ \Delta_x)^\star$$

Lemma

$$\begin{aligned}(p, q) \in \gamma(w) &\Leftrightarrow \exists u \in \Gamma(w) : p \xrightarrow{u} q \\ &\Leftrightarrow \exists u : \exists v \in \mathsf{suffixes}(w) : p \xrightarrow{u} q \wedge u \rightsquigarrow^\star \overline{v}v\end{aligned}$$

Histories

The set of histories is $G := \{r \in \mathcal{R}el(Q) \mid \exists w \in \mathbf{X}^\star : r = \gamma(w)\}$.

Closure Automaton

$\mathcal{cl}(\mathscr{A})$

$\mathcal{cl}(\mathscr{A}) := \langle Q \times G, \mathbf{X}, I \times \gamma(\epsilon), F \times G, \Delta' \rangle$ with transitions Δ' :

$$(q_1, \gamma(w)) \xrightarrow{x}_{\mathcal{cl}(\mathscr{A})} (q_2, \gamma(wx)) \text{ if } (q_1, q_2) \in \Delta_x \circ \gamma(wx)$$

Theorem

$$L(\mathcal{cl}(\mathscr{A})) = \mathcal{cl}(L(\mathscr{A}))$$

Size

$$\Delta' : \{((q_1, \gamma(w)), x, (q_2, \gamma(wx))) \mid (q_1, q_2) \in \Delta_x \circ \gamma(wx)\}$$

We can see that this construction produces a non-deterministic automaton of size at most $n \times 2^{n\times(n-1)}$.

Size

$$\Delta' : \{((q_1,\gamma(w)),x,(q_2,\gamma(wx))) \mid (q_1,q_2)\in\Delta_x\circ\gamma(wx)\}$$

We can see that this construction produces a non-deterministic automaton of size at most $n\times 2^{n\times(n-1)}$.

Furthermore, it can be easily determinized :

$$\delta' : ((Q_1,\gamma(w)),x)\mapsto(Q_1\cdot(\Delta_x\circ\gamma(wx)),\gamma(wx))$$

Size

$$\Delta' : \{((q_1, \gamma(w)), x, (q_2, \gamma(wx))) \mid (q_1, q_2) \in \Delta_x \circ \gamma(wx)\}$$

We can see that this construction produces a non-deterministic automaton of size at most $n \times 2^{n\times(n-1)}$.

Furthermore, it can be easily determinized :

$$\delta' : ((Q_1, \gamma(w)), x) \mapsto (Q_1 \cdot (\Delta_x \circ \gamma(wx)), \gamma(wx))$$

This deterministic automaton has at most $2^n \times 2^{n\times(n-1)} = 2^{n^2}$ states, which is significantly smaller than $2^{2^{n^2}}$, the size of the automaton from the original construction.

Plan

1. Introduction

2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra with converse
   - Reduction to an automaton problem

3. Closure of an automaton

4. The PSPACE algorithm.

5. Conclusion

Automaton equivalence

Let $\mathscr{A}$ and $\mathscr{B}$ be two deterministic automata over some alphabet Σ.

Theorem

$L(\mathscr{A}) \neq L(\mathscr{B}) \Leftrightarrow \exists w \in (L(\mathscr{A}) \ominus L(\mathscr{B})) : |w| \leqslant |\mathscr{A}| \times |\mathscr{B}|.$

Automaton equivalence

Let $\mathscr{A}$ and $\mathscr{B}$ be two deterministic automata over some alphabet Σ.

Theorem

$L(\mathscr{A}) \neq L(\mathscr{B}) \Leftrightarrow \exists w \in (L(\mathscr{A}) \ominus L(\mathscr{B})) : |w| \leqslant |\mathscr{A}| \times |\mathscr{B}|$.

```
input  : $\mathscr{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$
input  : $\mathscr{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$
output: A Boolean, saying whether or not $\mathscr{A}_1$ and $\mathscr{A}_2$ recognise the same language.
$N \leftarrow (|Q_1| \times |Q_2|)$ ;
$(p_1, p_2) \leftarrow (i_1, i_2)$ ;
while $N > 0$ do
    $N \leftarrow N - 1$;                          /* $N$ bounds the recursion depth */
    $f_1 \leftarrow$ is_in($p_1$, $T_1$);
    $f_2 \leftarrow$ is_in($p_2$, $T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\Sigma$) ;          /* Non-deterministic choice */
        $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
    else
        return false;      /* A difference appeared for some word, $L(\mathscr{A}_1) \neq L(\mathscr{A}_2)$ */
    end

end
return true;               /* There was no difference, $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ */
```

Automaton equivalence

Let $\mathscr{A}$ and $\mathscr{B}$ be two deterministic automata over some alphabet Σ.

Theorem

$$L(\mathscr{A}) \neq L(\mathscr{B}) \Leftrightarrow \exists w \in (L(\mathscr{A}) \ominus L(\mathscr{B})) : |w| \leqslant |\mathscr{A}| \times |\mathscr{B}|.$$

```
input  : $\mathscr{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$
input  : $\mathscr{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$
output: A Boolean, saying whether or not $\mathscr{A}_1$ and $\mathscr{A}_2$ recognise the same language.
$\boxed{N \leftarrow (|Q_1| \times |Q_2|)\,;}$
$(p_1, p_2) \leftarrow (i_1, i_2)$ ;
while $\boxed{N > 0}$ do
    $\boxed{N \leftarrow N - 1;}$                                /* $N$ bounds the recursion depth */
    $f_1 \leftarrow$ is_in($p_1, T_1$);
    $f_2 \leftarrow$ is_in($p_2, T_2$);
    if $f_1 = f_2$ then
        $\boxed{x \leftarrow \texttt{choose\_from}(\Sigma)\ ;}$                      /* Non-deterministic choice */
        $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
    else
        return false;          /* A difference appeared for some word, $L(\mathscr{A}_1) \neq L(\mathscr{A}_2)$ */
    end

end
return true;                          /* There was no difference, $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ */
```

Automaton equivalence

Let $\mathscr{A}$ and $\mathscr{B}$ be two deterministic automata over some alphabet Σ.

Theorem

$L(\mathscr{A}) \neq L(\mathscr{B}) \Leftrightarrow \exists w \in (L(\mathscr{A}) \ominus L(\mathscr{B})) : |w| \leqslant |\mathscr{A}| \times |\mathscr{B}|$.

input : $\mathscr{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$
input : $\mathscr{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$
output: A Boolean, saying whether or not $\mathscr{A}_1$ and $\mathscr{A}_2$ recognise the same language.

```
$N \leftarrow (|Q_1| \times |Q_2|)$ ;
$(p_1, p_2) \leftarrow (i_1, i_2)$ ;
while $N > 0$ do
    $N \leftarrow N - 1$;                                  /* $N$ bounds the recursion depth */
    $f_1 \leftarrow$ is_in($p_1, T_1$);
    $f_2 \leftarrow$ is_in($p_2, T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\Sigma$) ;                  /* Non-deterministic choice */
        $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
    else
        return false;        /* A difference appeared for some word, $L(\mathscr{A}_1) \neq L(\mathscr{A}_2)$ */
    end

end
return true;                 /* There was no difference, $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ */
```

Automaton equivalence

Let $\mathscr{A}$ and $\mathscr{B}$ be two deterministic automata over some alphabet Σ.

Theorem

$$L(\mathscr{A}) \neq L(\mathscr{B}) \Leftrightarrow \exists w \in (L(\mathscr{A}) \ominus L(\mathscr{B})) : |w| \leqslant |\mathscr{A}| \times |\mathscr{B}|.$$

```
input  : $\mathscr{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$
input  : $\mathscr{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$
output: A Boolean, saying whether or not $\mathscr{A}_1$ and $\mathscr{A}_2$ recognise the same language.

$N \leftarrow (|Q_1| \times |Q_2|)$ ;
$(p_1, p_2) \leftarrow (i_1, i_2)$ ;
while $N > 0$ do
    $N \leftarrow N - 1$;                                  /* $N$ bounds the recursion depth */
    $f_1 \leftarrow$ is_in($p_1, T_1$);
    $f_2 \leftarrow$ is_in($p_2, T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\Sigma$) ;               /* Non-deterministic choice */
        $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
    else
        return false;                /* A difference appeared for some word, $L(\mathscr{A}_1) \neq L(\mathscr{A}_2)$ */
    end

end
return true;                         /* There was no difference, $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ */
```

A PSPACE algorithm for KAC

```
input  : Two regular expressions with converse $e, f \in \mathcal{R}eg_X{}^{\vee}$
output: A Boolean, saying whether or not KAC $\vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$;
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

```
input  : Two regular expressions with converse $e, f \in \mathcal{R}eg_X{}^\vee$
output: A Boolean, saying whether or not KAC $\vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$;
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

```
input  : Two regular expressions with converse $e, f \in \mathcal{R}eg_X{}^{\vee}$
output: A Boolean, saying whether or not KAC $\vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$ ;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$ ;
$\boxed{N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})\,;}$
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$ ;
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

A PSpace algorithm for KAC

```
input  : Two regular expressions with converse $e, f \in \mathcal{R}eg_X{}^{\vee}$
output: A Boolean, saying whether or not KAC $\vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
$\boxed{((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}));}$
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

input : Two regular expressions with converse $e, f \in \mathcal{R}eg_X{}^{\vee}$

output: A Boolean, saying whether or not KAC $\vdash e = f$.

```
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$ ;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$ ;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$ ;
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$ ;
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $\boxed{(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star) ;}$
        $\boxed{(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2)) ;}$
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

Let's write n and m for the sizes of e and f.

```
input  : Two regular expressions with converse $e, f \in \mathcal{R}eg_X^\vee$
output: A Boolean, saying whether or not KAC $\vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$;
while $N > 0$ do
    $N \leftarrow N - 1$;
    $f_1 \leftarrow$ is_empty($P_1 \cap T_1$);
    $f_2 \leftarrow$ is_empty($P_2 \cap T_2$);
    if $f_1 = f_2$ then
        $x \leftarrow$choose_from($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

$\mathcal{O}(n + m)$

$\sim log\left(2^{n^2} \times 2^{m^2}\right) \sim \mathcal{O}\left(n^2 + m^2\right)$

$\sim log(n) + n^2 + log(m) + m^2$
$\sim \mathcal{O}\left(n^2 + m^2\right)$

$\mathcal{O}(log(n))$

$\mathcal{O}\left(n^2 + m^2\right)$

$\mathcal{O}\left(n^2 + m^2\right)$

So we get a space complexity $\mathcal{O}\left(n^2 + m^2\right)$.

So far

- New construction for deciding KAC.

So far

- New construction for deciding KAC.
- PSpace complexity.

So far

- New construction for deciding KAC.
- PSPACE complexity.
- Simpler correctness proofs.

So far

- New construction for deciding KAC.
- PSPACE complexity.
- Simpler correctness proofs.
- Toy implementation in OCAML of both constructions.

So far

- New construction for deciding KAC.
- PSPACE complexity.
- Simpler correctness proofs.
- Toy implementation in OCAML of both constructions.
- COQ proof of confluence of the relation $\rightsquigarrow$.

Further work

- Simpler proof of $\mathcal{cl}(\llbracket \mathbf{e} \rrbracket) = \mathcal{cl}(\llbracket \mathbf{f} \rrbracket) \Rightarrow \mathsf{KAC} \vdash e = f$ [(i)].

(i). Ésik, Z. and Bernátsky, L. (1995). Equational properties of Kleene algebras of relations with conversion.
Theoretical Computer Science, 137(2) :237–251

Further work

- Simpler proof of $cl(\llbracket \mathbf{e} \rrbracket) = cl(\llbracket \mathbf{f} \rrbracket) \Rightarrow \mathsf{KAC} \vdash e = f$ [(i)].
- Formalize in Coq.

[(i)]. Ésik, Z. and Bernátsky, L. (1995). Equational properties of Kleene algebras of relations with conversion.
Theoretical Computer Science, 137(2) :237–251

Further work

- Simpler proof of $cl(\llbracket \mathbf{e} \rrbracket) = cl(\llbracket \mathbf{f} \rrbracket) \Rightarrow \mathsf{KAC} \vdash e = f$ (i).
- Formalize in Coq.
- Other extensions of Kleene Algebra : Action Algebra ($\multimap$), Kleene Algebra with Intersection ($\wedge$)...

(i). Ésik, Z. and Bernátsky, L. (1995). Equational properties of Kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2) :237–251

That's it !

Thank you !

Plan

1. Introduction
2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra with converse
   - Reduction to an automaton problem
3. Closure of an automaton
4. The PSPACE algorithm.
5. Conclusion