

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

$$(x^* + y) \cdot z$$

$$(x^* \cdot z) + (y \cdot z)$$

Introduction

$$\forall S, \forall \sigma : \mathcal{Reg}_X \rightarrow \mathcal{Rel}(S)$$

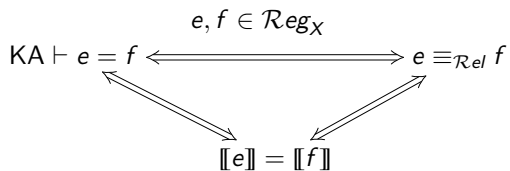
$$\begin{array}{ccc}
 (x^* + y) \cdot z & & (x^* \cdot z) + (y \cdot z) \\
 \downarrow & & \downarrow \\
 \sigma((x^* + y) \cdot z) & = & \sigma((x^* \cdot z) + (y \cdot z))
 \end{array}$$

Introduction

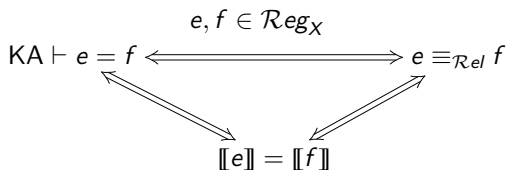
$$\begin{array}{ccc}
 (x^* + y) \cdot z & \equiv_{\mathcal{R}el} & (x^* \cdot z) + (y \cdot z) \\
 \downarrow & & \downarrow \\
 \sigma((x^* + y) \cdot z) & = & \sigma((x^* \cdot z) + (y \cdot z))
 \end{array}$$

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

Introduction



Introduction



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

Introduction

$$\begin{array}{ccc}
 & e, f \in \mathcal{R}eg_X & \\
 KA \vdash e = f & \longleftrightarrow & e \equiv_{\mathcal{R}el} f \\
 & \searrow \quad \swarrow & \\
 & \llbracket e \rrbracket = \llbracket f \rrbracket &
 \end{array}$$

$$\begin{array}{ccc}
 & e, f \in \mathcal{R}eg_X^\vee & \\
 KAC \vdash e = f & \longleftrightarrow & e \equiv_{\mathcal{R}el^\vee} f \\
 & \searrow \quad \swarrow & \\
 & cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket) &
 \end{array}$$

Introduction

$$e, f \in \text{Reg}_X^\vee$$

$$e \equiv_{\text{Reg}^\vee} f$$

$$cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket)$$

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

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

Regular expressions with converse

Regular expressions with converse over X

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

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 $e, f \in \mathcal{Reg}_X^\vee$:

$$e \equiv_{\mathcal{Rel}^\vee} f$$

means that

$$\forall S, \forall \sigma : X \rightarrow \mathcal{Rel}(S), \hat{\sigma}(e) = \hat{\sigma}(f).$$

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 :

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

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

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

$$a^{\vee\vee} = a \quad (4)$$

$$aa^{\vee}a \geq a \quad (5)$$

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 :

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

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

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

$$a^{\vee\vee} = a \quad (4)$$

$$aa^{\vee}a \geq a \quad (5)$$

From \mathcal{Reg}_X^\vee to \mathcal{Reg}_X

Let X be a finite alphabet. For $e \in \mathcal{Reg}_X$, we write $\llbracket e \rrbracket \subseteq X^*$ 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 :

$$\left\{ \begin{array}{l} (a + b)^\vee \mapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \mapsto b^\vee \cdot a^\vee \\ (a^*)^\vee \mapsto (a^\vee)^* \\ a^{\vee\vee} \mapsto a \end{array} \right.$$

From \mathcal{Reg}_X^\vee to \mathcal{Reg}_X

Let X be a finite alphabet. For $e \in \mathcal{Reg}_X$, we write $\llbracket e \rrbracket \subseteq X^*$ 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 :

$$\left\{ \begin{array}{l} (a + b)^\vee \mapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \mapsto b^\vee \cdot a^\vee \\ (a^*)^\vee \mapsto (a^\vee)^* \\ a^{\vee\vee} \mapsto a \\ x^\vee \mapsto x' \quad (x \in X) \\ x'^\vee \mapsto x \quad (x \in X) \end{array} \right.$$

From Reg_X^\vee to Reg_X

Let X be a finite alphabet. For $e \in \text{Reg}_X$, we write $\llbracket e \rrbracket \subseteq X^*$ 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 :

$$\left\{ \begin{array}{l} (a + b)^\vee \mapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \mapsto b^\vee \cdot a^\vee \\ (a^*)^\vee \mapsto (a^\vee)^* \\ a^{\vee\vee} \mapsto a \\ x^\vee \mapsto x' \quad (x \in X) \\ x'^\vee \mapsto x \quad (x \in X) \\ \mathbb{1}^\vee \mapsto \mathbb{1} \\ \mathbb{0}^\vee \mapsto \mathbb{0} \end{array} \right.$$

From Reg_X^\vee to Reg_X

Let X be a finite alphabet. For $e \in \text{Reg}_X$, we write $\llbracket e \rrbracket \subseteq X^*$ 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 :

$$\left\{ \begin{array}{l} (a + b)^\vee \mapsto a^\vee + b^\vee \\ (a \cdot b)^\vee \mapsto b^\vee \cdot a^\vee \\ (a^*)^\vee \mapsto (a^\vee)^* \\ a^{\vee\vee} \mapsto a \\ x^\vee \mapsto x' \quad (x \in X) \\ x'^\vee \mapsto x \quad (x \in X) \\ \mathbb{1}^\vee \mapsto \mathbb{1} \\ \mathbb{0}^\vee \mapsto \mathbb{0} \end{array} \right.$$

We get $e \in \text{Reg}_X$.

Is it enough ?

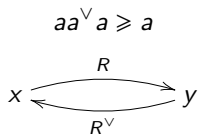
$$e, f \in \mathcal{Reg}_X : \quad e \equiv_{\mathcal{Rel}} f \quad \begin{array}{c} \Rightarrow \\ \Leftarrow \end{array} \quad \llbracket e \rrbracket = \llbracket f \rrbracket$$

Is it enough ?

$$e, f \in \mathcal{R}eg_X^V : \quad e \equiv_{\mathcal{R}el^V} f \quad \Leftarrow \quad \llbracket e \rrbracket = \llbracket f \rrbracket$$

Is it enough ?

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



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

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

Is it enough ?

$$e, f \in \mathcal{R}eg_X^\vee : \quad e \equiv_{\mathcal{R}el^N} f \quad \begin{array}{l} \Rightarrow \\ \Leftarrow \end{array} \quad cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket)$$

$$aa^\vee a \geq a$$

$$\begin{array}{ccc}
 & R & \\
 x & \xrightarrow{\quad} & y \\
 & \xleftarrow{\quad} & \\
 & R^\vee &
 \end{array}$$

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

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

Reduction relation and closure

Converse for words : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

$$\forall x \in X, \quad \bar{x} := x' \quad \left| \quad \bar{\epsilon} := \epsilon \right.$$

$$\forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \bar{x} \bar{w}.$$

Reduction relation and closure

Converse for words : \overline{w}

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

$$\forall x \in X, \quad \overline{x} := x' \quad \left| \quad \overline{\epsilon} := \epsilon \right.$$

$$\forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \overline{x} \overline{w}.$$

Reduction relation : $u \rightsquigarrow v$

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

Reduction relation and closure

Converse for words : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

$$\forall x \in X, \quad \bar{x} := x' \quad \left| \quad \bar{\epsilon} := \epsilon \right. \\ \forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \bar{x} \bar{w}.$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w \bar{w} w \cdot u_2 \rightsquigarrow u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

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

Reduction relation and closure

Converse for words : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

$$\forall x \in X, \quad \bar{x} := x' \quad \left| \quad \bar{\epsilon} := \epsilon \right. \\ \forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \bar{x} \bar{w}.$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w \bar{w} w \cdot u_2 \rightsquigarrow u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

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

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 : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

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

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w \bar{w} w \cdot u_2 \rightsquigarrow u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

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

Example :

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

Reduction relation and closure

Converse for words : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

$$\forall x \in X, \quad \bar{x} := x' \quad \left| \quad \bar{\epsilon} := \epsilon \right. \\ \forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \bar{x} \bar{w}.$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w \bar{w} w \cdot u_2 \rightsquigarrow u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

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

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 : \bar{w}

For a word $w \in X^*$, we define inductively \bar{w} :

$$\forall x \in X, \quad \bar{x} := x' \quad \left| \quad \bar{\epsilon} := \epsilon \right. \\ \forall x' \in X', \quad \overline{x'} := x \quad \left| \quad \overline{wx} := \bar{x} \bar{w}.$$

Reduction relation : $u \rightsquigarrow v$

$$\frac{}{u_1 \cdot w \bar{w} w \cdot u_2 \rightsquigarrow u_1 \cdot w \cdot u_2}$$

Closure of a language : $cl(L)$

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

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

a. Bloom, S. L., Ésik, Z., and Stefanescu, G. (1995). [Notes on equational theories of relations.](#)

Algebra Universalis, 33(1) :98–126

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

Plan

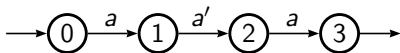
- 1 Introduction
- 2 From Kleene Algebra with Converse to regular languages
 - Kleene Algebra with converse
 - Reduction to an automaton problem
- 3 Closure of an automaton
- 4 The PSPACE algorithm.
- 5 Conclusion

Problem

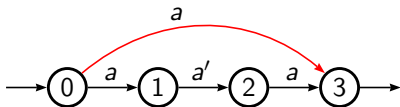
Input : an automaton \mathcal{A} over X

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

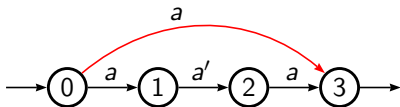
Intuition



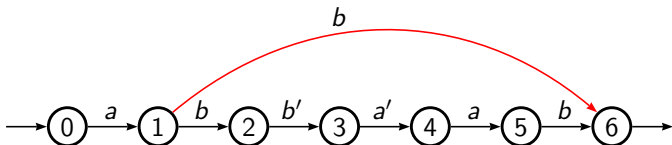
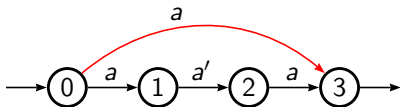
Intuition



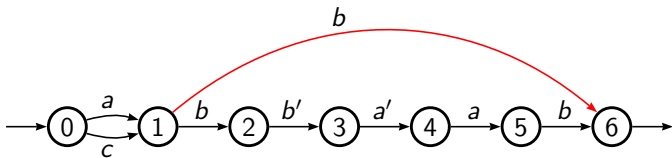
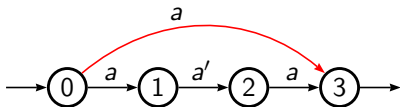
Intuition



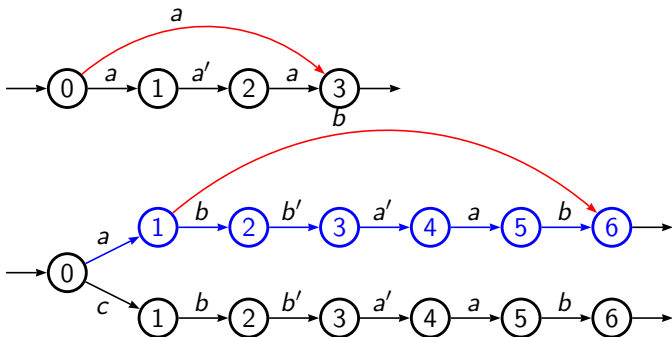
Intuition



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 :
 - ▶ $q_1 \xrightarrow{a} q_3$,
 - ▶ $H \xrightarrow{a} H'$,

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 :
 - ▶ $q_1 \xrightarrow{a} q_3$,
 - ▶ $H \xrightarrow{a} H'$,
 - ▶ and H' allows to jump from q_3 to q_2 .

$\Gamma(w)$ Definition : $\Gamma(w)$

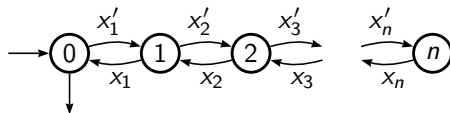
$$\Gamma(\epsilon) = \{\epsilon\}$$

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

Lemma

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

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



$\gamma(w)$

Consider an automaton $\mathcal{A} = \langle Q, A, I, T, \Delta \rangle$, we write

$$\Delta_x := \{(p, q) \mid p \xrightarrow{x} q \in \Delta\}.$$

Definition : $\gamma(w)$

$$\begin{aligned}\gamma(\epsilon) &= \text{Id}_Q \\ \gamma(wx) &= (\Delta_{x'} \circ \gamma(w) \circ \Delta_x)^*\end{aligned}$$

Lemma

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

Histories

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

Closure Automaton

$cl(\mathcal{A})$

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

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

Theorem

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

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 \mathcal{A} and \mathcal{B} be two deterministic automata over some alphabet Σ .

Theorem

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

Automaton equivalence

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

Theorem

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

input : $\mathcal{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$

input : $\mathcal{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$

output: A Boolean, saying whether or not \mathcal{A}_1 and \mathcal{A}_2 recognise the same language.

```

1  $N \leftarrow (|Q_1| \times |Q_2|)$ ;
2  $(p_1, p_2) \leftarrow (i_1, i_2)$ ;
3 while  $N > 0$  do
4    $N \leftarrow N - 1$ ;                                     /*  $N$  bounds the recursion depth */
5    $f_1 \leftarrow \text{is\_in}(p_1, T_1)$ ;
6    $f_2 \leftarrow \text{is\_in}(p_2, T_2)$ ;
7   if  $f_1 = f_2$  then
8      $x \leftarrow \text{choose\_from}(\Sigma)$ ;                       /* Non-deterministic choice */
9      $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
10  else
11    return false;                                       /* A difference appeared for some word,  $L(\mathcal{A}_1) \neq L(\mathcal{A}_2)$  */
12  end
13
14 end
15 return true;                                         /* There was no difference,  $L(\mathcal{A}_1) = L(\mathcal{A}_2)$  */

```

Automaton equivalence

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

Theorem

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

input : $\mathcal{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$

input : $\mathcal{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$

output: A Boolean, saying whether or not \mathcal{A}_1 and \mathcal{A}_2 recognise the same language.

```

1  $N \leftarrow (|Q_1| \times |Q_2|);$ 
2  $(p_1, p_2) \leftarrow (i_1, i_2);$ 
3 while  $(N > 0)$  do
4    $N \leftarrow N - 1;$                                      /* N bounds the recursion depth */
5    $f_1 \leftarrow \text{is\_in}(p_1, T_1);$ 
6    $f_2 \leftarrow \text{is\_in}(p_2, T_2);$ 
7   if  $f_1 = f_2$  then
8      $x \leftarrow \text{choose\_from}(\Sigma);$                    /* Non-deterministic choice */
9      $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x));$ 
10  else
11    return false;                                       /* A difference appeared for some word,  $L(\mathcal{A}_1) \neq L(\mathcal{A}_2)$  */
12  end
13
14 end
15 return true;                                         /* There was no difference,  $L(\mathcal{A}_1) = L(\mathcal{A}_2)$  */

```

Automaton equivalence

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

Theorem

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

input : $\mathcal{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$

input : $\mathcal{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$

output: A Boolean, saying whether or not \mathcal{A}_1 and \mathcal{A}_2 recognise the same language.

```

1  N ← (|Q1| × |Q2|);
2  (p1, p2) ← (i1, i2);
3  while N > 0 do
4      N ← N - 1;                               /* N bounds the recursion depth */
5      f1 ← is_in(p1, T1);
6      f2 ← is_in(p2, T2);
7      if f1 = f2 then
8          x ← choose_from(Σ);                   /* Non-deterministic choice */
9          (p1, p2) ← (δ1(p1, x), δ2(p2, x));
10     else
11         return false;                         /* A difference appeared for some word, L(ℳ1) ≠ L(ℳ2) */
12     end
13
14 end
15 return true;                                 /* There was no difference, L(ℳ1) = L(ℳ2) */

```


Automaton equivalence

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

Theorem

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

input : $\mathcal{A}_1 = \langle Q_1, \Sigma, i_1, T_1, \delta_1 \rangle$

input : $\mathcal{A}_2 = \langle Q_2, \Sigma, i_2, T_2, \delta_2 \rangle$

output: A Boolean, saying whether or not \mathcal{A}_1 and \mathcal{A}_2 recognise the same language.

```

1  $N \leftarrow (|Q_1| \times |Q_2|)$ ;
2  $(p_1, p_2) \leftarrow (i_1, i_2)$ ;
3 while  $N > 0$  do
4    $N \leftarrow N - 1$ ;                                /*  $N$  bounds the recursion depth */
5    $f_1 \leftarrow \text{is\_in}(p_1, T_1)$ ;
6    $f_2 \leftarrow \text{is\_in}(p_2, T_2)$ ;
7   if  $f_1 = f_2$  then
8      $x \leftarrow \text{choose\_from}(\Sigma)$ ;                /* Non-deterministic choice */
9      $(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))$ ;
10  else
11    return false;                                  /* A difference appeared for some word,  $L(\mathcal{A}_1) \neq L(\mathcal{A}_2)$  */
12  end
13
14 end
15 return true;                                       /* There was no difference,  $L(\mathcal{A}_1) = L(\mathcal{A}_2)$  */

```

A PSPACE algorithm for KAC

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

- 1 $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, l_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket e \rrbracket$;
- 2 $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, l_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket f \rrbracket$;
- 3 $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
- 4 $((P_1, R_1), (P_2, R_2)) \leftarrow ((l_1, \text{Id}_{Q_1}), (l_2, \text{Id}_{Q_1}))$;
- 5 **while** $N > 0$ **do**
- 6 $N \leftarrow N - 1$;
- 7 $f_1 \leftarrow \text{is_empty}(P_1 \cap T_1)$;
- 8 $f_2 \leftarrow \text{is_empty}(P_2 \cap T_2)$;
- 9 **if** $f_1 = f_2$ **then**
- 10 $x \leftarrow \text{choose_from}(\mathbf{X})$;
- 11 $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$;
- 12 $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
- 13 **else**
- 14 **return** false
- 15 **end**
- 16 **end**
- 17 **return** true

A PSPACE algorithm for KAC

input : Two regular expressions with converse $e, f \in \text{Reg}_X^\vee$

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

```

1  $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, l_1, T_1, \Delta_1 \rangle \leftarrow$  Glushkov' automaton recognising  $\llbracket e \rrbracket$ ;
2  $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, l_2, T_2, \Delta_2 \rangle \leftarrow$  Glushkov' automaton recognising  $\llbracket f \rrbracket$ ;
3  $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$ ;
4  $((P_1, R_1), (P_2, R_2)) \leftarrow ((l_1, \text{Id}_{Q_1}), (l_2, \text{Id}_{Q_2}))$ ;
5 while  $N > 0$  do
6    $N \leftarrow N - 1$ ;
7    $f_1 \leftarrow \text{is\_empty}(P_1 \cap T_1)$ ;
8    $f_2 \leftarrow \text{is\_empty}(P_2 \cap T_2)$ ;
9   if  $f_1 = f_2$  then
10     $x \leftarrow \text{choose\_from}(\mathbf{X})$ ;
11     $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$ ;
12     $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
13  else
14    return false
15  end
16 end
17 return true

```

A PSPACE algorithm for KAC

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

- 1 $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, l_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket e \rrbracket$;
- 2 $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, l_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket f \rrbracket$;
- 3 $N \leftarrow (2^{(|e|+1)^c} \times 2^{(|f|+1)^c})$;
- 4 $((P_1, R_1), (P_2, R_2)) \leftarrow ((l_1, \text{Id}_{Q_1}), (l_2, \text{Id}_{Q_1}))$;
- 5 **while** $N > 0$ **do**
- 6 $N \leftarrow N - 1$;
- 7 $f_1 \leftarrow \text{is_empty}(P_1 \cap T_1)$;
- 8 $f_2 \leftarrow \text{is_empty}(P_2 \cap T_2)$;
- 9 **if** $f_1 = f_2$ **then**
- 10 $x \leftarrow \text{choose_from}(\mathbf{X})$;
- 11 $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$;
- 12 $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
- 13 **else**
- 14 **return** false
- 15 **end**
- 16 **end**
- 17 **return** true

A PSPACE algorithm for KAC

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

- 1 $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket e \rrbracket$;
- 2 $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket f \rrbracket$;
- 3 $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
- 4 $((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \text{Id}_{Q_1}), (I_2, \text{Id}_{Q_2}))$;
- 5 **while** $N > 0$ **do**
- 6 $N \leftarrow N - 1$;
- 7 $f_1 \leftarrow \text{is_empty}(P_1 \cap T_1)$;
- 8 $f_2 \leftarrow \text{is_empty}(P_2 \cap T_2)$;
- 9 **if** $f_1 = f_2$ **then**
- 10 $x \leftarrow \text{choose_from}(\mathbf{X})$;
- 11 $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$;
- 12 $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
- 13 **else**
- 14 **return** false
- 15 **end**
- 16 **end**
- 17 **return** true

A PSPACE algorithm for KAC

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

- 1 $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket e \rrbracket$;
- 2 $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $\llbracket f \rrbracket$;
- 3 $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$;
- 4 $((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \text{Id}_{Q_1}), (I_2, \text{Id}_{Q_2}))$;
- 5 **while** $N > 0$ **do**
- 6 $N \leftarrow N - 1$;
- 7 $f_1 \leftarrow \text{is_empty}(P_1 \cap T_1)$;
- 8 $f_2 \leftarrow \text{is_empty}(P_2 \cap T_2)$;
- 9 **if** $f_1 = f_2$ **then**
- 10 $x \leftarrow \text{choose_from}(\mathbf{X})$;
- 11 $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$;
- 12 $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
- 13 **else**
- 14 **return** false
- 15 **end**
- 16 **end**
- 17 **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 $e, f \in \text{Reg}_X^\vee$
output: A Boolean, saying whether or not $\text{KAC} \vdash e = f$.

```

1  $\mathcal{A}_1 = \langle Q_1, \mathbf{X}, l_1, T_1, \Delta_1 \rangle \leftarrow$  Glushkov' automaton recognising  $\llbracket e \rrbracket$ ;
2  $\mathcal{A}_2 = \langle Q_2, \mathbf{X}, l_2, T_2, \Delta_2 \rangle \leftarrow$  Glushkov' automaton recognising  $\llbracket f \rrbracket$ ;
3  $N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$ ;
4  $((P_1, R_1), (P_2, R_2)) \leftarrow ((l_1, \text{Id}_{Q_1}), (l_2, \text{Id}_{Q_1}))$ ;
5 while  $N > 0$  do
6    $N \leftarrow N - 1$ ;
7    $f_1 \leftarrow \text{is\_empty}(P_1 \cap T_1)$ ;
8    $f_2 \leftarrow \text{is\_empty}(P_2 \cap T_2)$ ;
9   if  $f_1 = f_2$  then
10     $x \leftarrow \text{choose\_from}(\mathbf{X})$ ;
11     $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^*, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^*)$ ;
12     $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
13  else
14    return false
15  end
16 end
17 return true

```

Complexity annotations:

- $\mathcal{O}(n + m)$ (lines 1-2)
- $\sim \log(2^{n^2} \times 2^{m^2}) \sim \mathcal{O}(n^2 + m^2)$ (line 3)
- $\sim \log(n) + n^2 + \log(m) + m^2 \sim \mathcal{O}(n^2 + m^2)$ (line 4)
- $\mathcal{O}(\log(n))$ (line 8)
- $\mathcal{O}(n^2 + m^2)$ (line 11)
- $\mathcal{O}(n^2 + m^2)$ (line 12)

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

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 $cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket) \Rightarrow \text{KAC} \vdash e = f^{(i)}$.

(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 $cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket) \Rightarrow \text{KAC} \vdash e = f^{(i)}$.
- 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 $cl(\llbracket e \rrbracket) = cl(\llbracket f \rrbracket) \Rightarrow \text{KAC} \vdash e = f^{(i)}$.
- Formalize in Coq.
- Other extensions of Kleene Algebra : Action Algebra (\rightarrow), 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