Deciding Kleene Algebra with converse is PSPACE-*complete*

Talk at the PACE meeting

Paul Brunet & Damien Pous

ENS de Lyon

February 9^{th}, 2014

Introduction

Kleene Algebra (i) : Abstraction for proving the equivalence of regular expressions.

(i). Conway, J. H. (1971). *Regular algebra and finite machines.* Chapman and Hall Mathematics Series

Introduction

Kleene Algebra (i) : Abstraction for proving the equivalence of regular expressions.
The equivalence is PSPACE-*complete*.

(i). Conway, J. H. (1971). *Regular algebra and finite machines.*
Chapman and Hall Mathematics Series

Introduction

Kleene Algebra (i) : Abstraction for proving the equivalence of regular expressions.
The equivalence is PSpace-*complete*.

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

(i). Conway, J. H. (1971). *Regular algebra and finite machines.* Chapman and Hall Mathematics Series

Introduction

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

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

Introduction

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

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

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

$$\begin{tikzcd}
\mathrm{KAC} \vdash e = f \arrow[rr, Leftrightarrow] \arrow[dr, Leftrightarrow] & & e \equiv_{\mathcal{R}el^{\vee}} f \\
& \mathscr{cl}(\llbracket \mathbf{e} \rrbracket) = \mathscr{cl}(\llbracket \mathbf{f} \rrbracket) \arrow[ur, Leftrightarrow] &
\end{tikzcd}$$

Introduction

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

$$\begin{tikzcd}
\mathrm{KAC} \vdash e = f \arrow[rr, Leftrightarrow, "{e, f \in \mathcal{R}eg_X{}^{\vee}}"] & & e \equiv_{\mathcal{R}el^{\vee}} f \\
& \mathcal{cl}\left(\llbracket \mathbf{e} \rrbracket\right) = \mathcal{cl}\left(\llbracket \mathbf{f} \rrbracket\right) \arrow[ul, Leftrightarrow] \arrow[ur, Leftrightarrow] &
\end{tikzcd}$$

Introduction

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

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

Introduction

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

$$\begin{tikzcd} & e \equiv_{\mathcal{R}el^{\vee}} f \arrow[dl, Leftrightarrow] \\ c\ell(\llbracket \mathbf{e} \rrbracket) = c\ell(\llbracket \mathbf{f} \rrbracket) & \end{tikzcd}$$

Plan

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

Plan

1. Introduction

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

3. Closure of an automaton

4. The PSPACE algorithm.

Regular expressions with converse

Regular expressions with converse over X

Let X be a finite set, the set of regular expressions over X (written $\mathcal{R}eg_X{}^\vee$) are obtained with the grammar :

$$e, f ::= \mathbb{0} | \mathbb{1} | x \in X | e + f | e \cdot f | e^* | e^\vee$$

Regular expressions with converse

Regular expressions with converse over X

Let X be a finite set, the set of regular expressions over X (written $\mathcal{R}eg_X{}^{\vee}$) are obtained with the grammar :

$$e, f ::= \mathbb{0} | \mathbb{1} | x \in X | e + f | e \cdot f | e^* | e^{\vee}$$

A relational interpretation of regular expressions with converse over X can be specified by a domain S and a map

$$\sigma : X \longrightarrow \mathcal{P}\left(S^2\right)$$

We will write

$$\hat{\sigma} : \mathcal{R}eg_X{}^{\vee} \longrightarrow \mathcal{P}\left(S^2\right)$$

for the unique morphism equal to σ on X.

Relational equivalence

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

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

means that

$$\forall S, \forall \sigma : X \to \mathcal{P}\left(S^2\right), \hat{\sigma}(e) = \hat{\sigma}(f).$$

A hint from the equational theory.

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

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

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

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

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

A hint from the equational theory.

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

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

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

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

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

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 $[\![e]\!] \subseteq X^*$ for the *language denoted by e*.

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

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 $[\![e]\!] \subseteq X^*$ for the *language denoted by* e.

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

1. We see equations (1)-(4) as rewriting rules :

$$\begin{aligned}
(a+b)^\vee &\longmapsto a^\vee + b^\vee\\
(a \cdot b)^\vee &\longmapsto b^\vee \cdot a^\vee\\
(a^*)^\vee &\longmapsto (a^\vee)^*\\
a^{\vee^\vee} &\longmapsto a\\
\mathbb{1}^\vee &\longmapsto \mathbb{1}\\
\mathbb{0}^\vee &\longmapsto \mathbb{0}
\end{aligned}$$

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 $[\![e]\!] \subseteq X^*$ for the *language denoted by* e.

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

1. We see equations (1)-(4) as rewriting rules :

$$\begin{array}{rl}
(a+b)^\vee & \longmapsto a^\vee + b^\vee \\
(a \cdot b)^\vee & \longmapsto b^\vee \cdot a^\vee \\
(a^*)^\vee & \longmapsto (a^\vee)^* \\
a^{\vee\vee} & \longmapsto a \\
\mathbb{1}^\vee & \longmapsto \mathbb{1} \\
\mathbb{0}^\vee & \longmapsto \mathbb{0}
\end{array}$$

2. We substitute $x^\vee$ with x' in the result. We get $\mathbf{e} \in \mathcal{R}eg_{\mathbf{X}}$.

Reduction relation

$$a \leqslant a a^{\vee} a$$

$\overline{w}$

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

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

Reduction relation

$$a \leqslant a a^{\vee} a$$

$\overline{w}$

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

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

$u \rightsquigarrow v$

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

Example :

$abbabb'a'abbaa'$

Reduction relation

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

$\overline{w}$

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

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

$u \rightsquigarrow v$

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

Example :

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

Reduction relation

$$a \leqslant a a^{\vee} a$$

$\overline{w}$

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

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

$u \rightsquigarrow v$

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

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

$$a \leqslant a a^{\vee} a$$

$\overline{w}$

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

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

$u \rightsquigarrow v$

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

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

$\mathcal{cl}(L)$

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

Closure

$cl(L)$

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

Theorem [a]

---

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

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

Plan

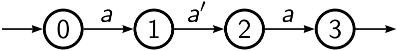
1. Introduction
2. From Kleene Algebra with Converse to regular languages
   - Kleene Algebra wih converse
   - Reducton to an automaton problem
3. Closure of an automaton
4. The PSpace algorithm.

Problem

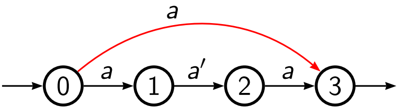
Input : an automaton $\mathscr{A}$

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

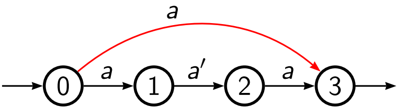
Intuition

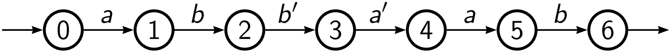


Intuition

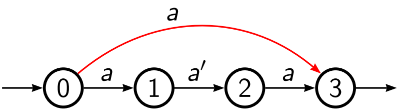


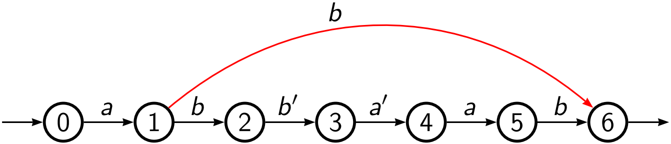
Intuition



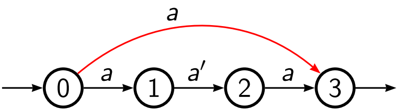


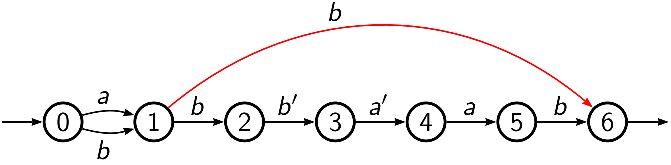
Intuition



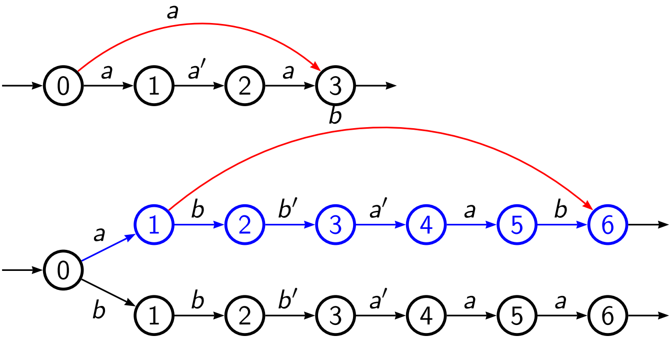


Intuition





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
  - and some history.

$$\text{If} : q_0 \xrightarrow{u} q_1 \xrightarrow{x} q_3 \xrightarrow{w} q_2$$

$$\text{With} : \exists u_2 \in \text{suffixes}(ux) : w \rightsquigarrow^* \overline{u}_2 u_2$$

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.

$$\text{If : } q_0 \xrightarrow{u} q_1 \xrightarrow{x} q_3 \xrightarrow{w} q_2$$

$$\text{With : } \exists u_2 \in \text{suffixes}(ux) : w \rightsquigarrow^* \overline{u}_2 u_2$$

$$\text{Meaning : } uxw \rightsquigarrow^* ux\overline{u}_2 u_2 = u_1 u_2 \overline{u}_2 u_2 \rightsquigarrow u_1 u_2 = ux$$

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.

$$\text{If : } q_0 \xrightarrow{u} q_1 \xrightarrow{x} q_3 \xrightarrow{w} q_2$$

$$\text{With : } \exists u_2 \in \text{suffixes}(ux) : w \rightsquigarrow^* \overline{u}_2 u_2$$

$$\text{Meaning : } uxw \rightsquigarrow^* ux\overline{u}_2 u_2 = u_1 u_2 \overline{u}_2 u_2 \rightsquigarrow u_1 u_2 = ux$$

$$\text{Then : } (q_0, \gamma(\epsilon)) \xrightarrow{u} (q_1, \gamma(u)) \xrightarrow{x} (q_2, \gamma(ux))$$

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.

$$\text{If} : q_0 \xrightarrow{u} q_1 \xrightarrow{x} q_3 \xrightarrow{w} q_2$$

$$\text{With} : \exists u_2 \in \text{suffixes}(ux) : w \rightsquigarrow^* \overline{u}_2 u_2$$

$$\text{Meaning} : uxw \rightsquigarrow^* ux\overline{u}_2 u_2 = u_1 u_2 \overline{u}_2 u_2 \rightsquigarrow u_1 u_2 = ux$$

$$\text{Then} : (q_0, \gamma(\epsilon)) \xrightarrow{u} (q_1, \gamma(u)) \xrightarrow{x} (q_2, \gamma(ux))$$

$\Gamma(w)$

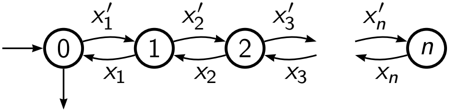
Definition : $\Gamma(w)$

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

Lemma

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

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



$\gamma(w)$

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

Definition : $\gamma(w)$

$$\gamma(\epsilon) = \mathrm{Id}_Q$$
$$\gamma(wx) = (\Delta_{x'} \cdot \gamma(w) \cdot \Delta_x)^*$$

Lemma

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

Histories

The set of histories is $G \coloneqq \left\{r \in \mathcal{P}\left(Q^2\right) \mid \exists w \in \mathbf{X}^* : r = \gamma(w)\right\}$.

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

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

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

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

$$(p, \gamma(u)) \xrightarrow{x} (q, \gamma(ux)) \triangleq \begin{array}{l} \exists r \in Q \\ \exists w \in \Gamma(ux) \end{array} \;:\; p \xrightarrow{x} r \xrightarrow{w} q$$

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

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

$$(p, \gamma(u)) \xrightarrow{x} (q, \gamma(ux)) \triangleq \begin{array}{l} \exists r \in Q \\ \exists v \in \text{suffixes}(ux) \\ \exists w \rightsquigarrow^* \overline{v} v \end{array} \ : \ p \xrightarrow{x} r \xrightarrow{w} q$$

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 $n \times 2^{n\times(n-1)}$.

Futhurmore, 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 $n \times 2^{n \times (n-1)}$.

Futhurmore, 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 significantely 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 wih converse
   - Reducton to an automaton problem
3. Closure of an automaton
4. The PSpace algorithm.

Automaton equivalence

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

Theorem

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

Automaton equivalence

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

Theorem

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

```
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
        $\boxed{x \leftarrow \texttt{random}(\Sigma)\ ;}$                /* 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

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

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 $N > 0$ 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
        $x \leftarrow$random($\Sigma$) ;                /* 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

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

```
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.

$N \leftarrow (|Q_1| \times |Q_2|)$;
$\boxed{(p_1, p_2) \leftarrow (i_1, i_2);}$
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
        $x \leftarrow$random($\Sigma$) ;                          /* 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

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

```
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
        $x \leftarrow$random($\Sigma$) ;                   /* 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

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

```
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
    $\boxed{N \leftarrow N - 1;}$                          /* $N$ 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
        $x \leftarrow$random($\Sigma$) ;                     /* 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

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

```
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
        $x \leftarrow$random($\Sigma$) ;                /* 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

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

```
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
        $\boxed{x \leftarrow \texttt{random}(\Sigma)\ ;}$          /* 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

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

```
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
        $x \leftarrow$random($\Sigma$) ;                        /* Non-deterministic choice */
        $\boxed{(p_1, p_2) \leftarrow (\delta_1(p_1, x), \delta_2(p_2, x))\,;}$
    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

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

```
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
        $x \leftarrow$random($\Sigma$) ;                  /* 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

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

```
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
        $x \leftarrow$random($\Sigma$) ;                      /* 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 $\text{KAC} \vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ 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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^{\star}, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^{\star})$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

A PSpace algorithm for KAC

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

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
        $x \leftarrow$random$(\mathbf{X})$;
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

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

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 $[\![\mathbf{e}]\!]$ ;  $\leftarrow \boxed{\mathcal{O}(n+m)}$
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ 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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

A PSPACE algorithm for KAC

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

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

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

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^{\star}, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^{\star})$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

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

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

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 $\mathrm{KAC} \vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ 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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

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

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

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

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

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

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

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

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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$ ;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$ ;
    else
        return false
    end
end
return true
```

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

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

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

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

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

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

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

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

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

n^2

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 $\mathrm{KAC} \vdash e = f$.
$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ 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
        $x \leftarrow$random($\mathbf{X}$);
        $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^{\star}, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^{\star})$;
        $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
    else
        return false
    end
end
return true
```

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

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

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

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

$2 \times m^2$

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

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

Plan

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