From regular expressions to automata

Marcello Bonsangue[1,2] Jan Rutten[1,3] Alexandra Silva[1]

[1]Centrum voor Wiskunde en Informatica
[2]LIACS - Leiden University
[3]Vrije Universiteit Amsterdam

January 2009

Motivation

Context: Regular expressions
Goal: Decide $r_1 = r_2$.
Usual approach:

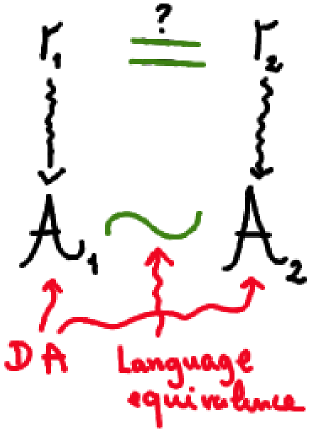
Motivation

Context: Regular expressions
Goal: Decide $r_1 = r_2$.
Usual approach:

$$\begin{tikzcd} r_1 \arrow[r, phantom, "\stackrel{?}{=}"] \arrow[d, squiggly] & r_2 \arrow[d, squiggly] \\ \mathcal{A}_1 \arrow[r, phantom, "\sim"] & \mathcal{A}_2 \end{tikzcd}$$

Motivation

Context: Regular expressions
Goal: Decide $r_1 = r_2$.
Usual approach:



For Regular Expressions to Deterministic automata

- Direct method : Brzozowski derivatives

$$(ab + b)^* ba$$

Problems:

1. Comparing derivatives is very expensive.
2. Method does not scale so well (cf. Circ/KAT)

For Regular Expressions to Deterministic automata

- Direct method : Brzozowski derivatives

$$(ab+b)^*ba$$

$$\begin{tikzcd}
\arrow[r] & \boxed{(ab+b)^*ba} \arrow[r, "a"] \arrow[d, "b"] & \boxed{(ab+b)^*ba+a} \\
& \boxed{b(ab+b)^*ba} &
\end{tikzcd}$$

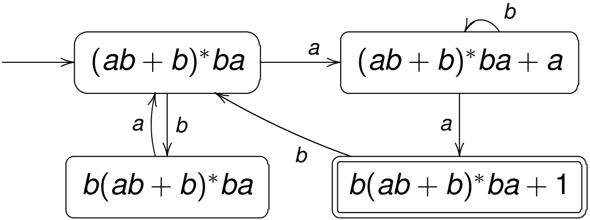
Problems:

1. Comparing derivatives is very expensive.
2. Method does not scale so well (cf. Circ/KAT)

For Regular Expressions to Deterministic automata

- Direct method : Brzozowski derivatives

$$(ab + b)^* ba$$



Problems:

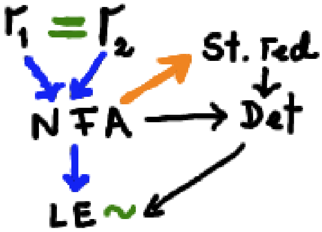
1. Comparing derivatives is very expensive.
2. Method does not scale so well (cf. Circ/KAT)

More efficient algorithms

1. Partial derivative (Antimirov)
2. Continuation automaton (Berry-Sethi)
3. Position automaton (Berry-Sethi, Glushkov, McNaughton-Yamada)

More efficient algorithms

1. Partial derivative (Antimirov)
2. Continuation automaton (Berry-Sethi)
3. Position automaton (Berry-Sethi, Glushkov, McNaughton-Yamada)



Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \rightarrow (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \to (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \to (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \rightarrow (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \rightarrow (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Continuation/Position automaton

- Basic idea: Assume that all occurrences of letters are different.

$$(ab + b)^* ba \to (a_1 b_2 + b_3)^* b_4 a_5$$

- Let the magic begin: example in the board.
- Key: The states are known from the beginning!

Theorem

Let all symbols in E be distinct. Given any symbol a, for all strings w, $(wa)^{-1}E$ is either 0 or unique modulo ACI.

- Nice property: all the transitions entering a state have the same label.
- Not so nice: the continuations need to be computed explicitly.

Position automaton

Definition

Let E be a regular expression and $\overline{E}$ the corresponding marked expression.

$$\begin{array}{lcl} first(E) & = & \{i \mid a_i w \in L(\overline{E})\} \\ follow(E,i) & = & \{j \mid u a_i a_j v \in L(\overline{E})\} \\ last(E) & = & \{i \mid w a_i \in L(\overline{E})\} \end{array}$$

The position automaton for E is defined as :

$$\mathcal{A}_{\mathbf{pos}}(E) = (pos(E), \Sigma, \delta_{\mathbf{pos}}, 0, last(E))$$

where

$$\delta_{\mathbf{pos}} = \{(i, a, j) \mid j \in follow(E, i), a = \overline{a_j}\}$$

Remark: Berry-Sethi provide an efficient algorithm to compute *follow*. Example in the board.

Position automaton

Definition

Let E be a regular expression and $\overline{E}$ the corresponding marked expression.

$$\begin{array}{lcl} \mathit{first}(E) & = & \{i \mid a_i w \in L(\overline{E})\} \\ \mathit{follow}(E,i) & = & \{j \mid u a_i a_j v \in L(\overline{E})\} \\ \mathit{last}(E) & = & \{i \mid w a_i \in L(\overline{E})\} \end{array}$$

The position automaton for E is defined as :

$$\mathcal{A}_{\mathbf{pos}}(E) = (\mathit{pos}(E), \Sigma, \delta_{\mathbf{pos}}, 0, \mathit{last}(E))$$

where

$$\delta_{\mathbf{pos}} = \{(i, a, j) \mid j \in \mathit{follow}(E, i), a = \overline{a_j}\}$$

Remark: Berry-Sethi provide an efficient algorithm to compute *follow*. Example in the board.

Language equivalence for NFA

Definition ($Q_1 \equiv Q_2$)

Let $\mathcal{A}_1 = (\Sigma, S_1, I_1, \delta_1, F_1)$ and $\mathcal{A}_1 = (\Sigma, S_2, I_2, \delta_2, F_2)$ be NFA's.
For $Q_1 \subseteq S_1$ and $Q_2 \subseteq S_2$, $Q_1 \equiv Q_2$ iff

1. $Q_1 \cap F_1 \neq \emptyset \Leftrightarrow Q_2 \cap F_2 \neq \emptyset$
2. $\delta_1[Q_1](a) \equiv \delta_2[Q_2](a)$, for all $a \in \Sigma$.

Theorem

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2) \Leftrightarrow I_1 \equiv I_2$$

Worst case = determinization + bisimilarity

Language equivalence for NFA

Definition ($Q_1 \equiv Q_2$)

Let $\mathcal{A}_1 = (\Sigma, S_1, I_1, \delta_1, F_1)$ and $\mathcal{A}_1 = (\Sigma, S_2, I_2, \delta_2, F_2)$ be NFA's.
For $Q_1 \subseteq S_1$ and $Q_2 \subseteq S_2$, $Q_1 \equiv Q_2$ iff

1. $Q_1 \cap F_1 \neq \emptyset \Leftrightarrow Q_2 \cap F_2 \neq \emptyset$
2. $\delta_1[Q_1](a) \equiv \delta_2[Q_2](a)$, for all $a \in \Sigma$.

Theorem

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2) \Leftrightarrow I_1 \equiv I_2$$

Worst case = determinization + bisimilarity

Language equivalence for NFA

Definition ($Q_1 \equiv Q_2$)

Let $\mathcal{A}_1 = (\Sigma, S_1, I_1, \delta_1, F_1)$ and $\mathcal{A}_1 = (\Sigma, S_2, I_2, \delta_2, F_2)$ be NFA's.
For $Q_1 \subseteq S_1$ and $Q_2 \subseteq S_2$, $Q_1 \equiv Q_2$ iff

1. $Q_1 \cap F_1 \neq \emptyset \Leftrightarrow Q_2 \cap F_2 \neq \emptyset$
2. $\delta_1[Q_1](a) \equiv \delta_2[Q_2](a)$, for all $a \in \Sigma$.

Theorem

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2) \Leftrightarrow I_1 \equiv I_2$$

Worst case = determinization + bisimilarity

Language equivalence for NFA

Definition ($Q_1 \equiv Q_2$)

Let $\mathcal{A}_1 = (\Sigma, S_1, I_1, \delta_1, F_1)$ and $\mathcal{A}_1 = (\Sigma, S_2, I_2, \delta_2, F_2)$ be NFA's.
For $Q_1 \subseteq S_1$ and $Q_2 \subseteq S_2$, $Q_1 \equiv Q_2$ iff

1. $Q_1 \cap F_1 \neq \emptyset \Leftrightarrow Q_2 \cap F_2 \neq \emptyset$
2. $\delta_1[Q_1](a) \equiv \delta_2[Q_2](a)$, for all $a \in \Sigma$.

Theorem

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2) \Leftrightarrow I_1 \equiv I_2$$

Worst case = determinization + bisimilarity

Language equivalence for NFA

Definition ($Q_1 \equiv Q_2$)

Let $\mathcal{A}_1 = (\Sigma, S_1, I_1, \delta_1, F_1)$ and $\mathcal{A}_1 = (\Sigma, S_2, I_2, \delta_2, F_2)$ be NFA's.
For $Q_1 \subseteq S_1$ and $Q_2 \subseteq S_2$, $Q_1 \equiv Q_2$ iff

1. $Q_1 \cap F_1 \neq \emptyset \Leftrightarrow Q_2 \cap F_2 \neq \emptyset$
2. $\delta_1[Q_1](a) \equiv \delta_2[Q_2](a)$, for all $a \in \Sigma$.

Theorem

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2) \Leftrightarrow I_1 \equiv I_2$$

Worst case = determinization + bisimilarity

Conclusions

Can we implement this algorithms in CIRC?
Can we extend them to KAT?