Learning Restricted Restarting Automata

Presentation for the ABCD workshop in Prague, March 27 – 29, 2009

Peter Černo

About the presentation

- **PART I**: We present a specialized program which allows an easy design and testing of *restarting automata* and provides specialized tools for learning finite automata and defining languages.

About the presentation

- **PART I**: We present a specialized program which allows an easy design and testing of *restarting automata* and provides specialized tools for learning finite automata and defining languages.
- **PART II**: We introduce two restricted models of *restarting automata*: RA_{CL} and $RA_{\Delta CL}$ together with their properties and limitations.

About the presentation

- **PART I**: We present a specialized program which allows an easy design and testing of *restarting automata* and provides specialized tools for learning finite automata and defining languages.
- **PART II**: We introduce two restricted models of *restarting automata*: RA_{CL} and $RA_{\Delta CL}$ together with their properties and limitations.
- **PART III**: We demonstrate learning of these restricted models by another specialized program.

About the presentation

- **PART I**: We present a specialized program which allows an easy design and testing of *restarting automata* and provides specialized tools for learning finite automata and defining languages.
- **PART II**: We introduce two restricted models of *restarting automata*: RA_{CL} and $RA_{\Delta CL}$ together with their properties and limitations.
- **PART III**: We demonstrate learning of these restricted models by another specialized program.
- **PART IV**: We give a list of some open problems and topics for future investigations.

Restarting Automaton

- Is a system $M = (\Sigma, \Gamma, I)$ where:
- Σ is an *input alphabet*, Γ is a *working alphabet*
- I is a finite set of *meta-instructions*:
  - *Rewriting* meta-instruction $(E_L, x \to y, E_R)$, where $x, y \in \Gamma^*$ such that $|x| > |y|$, and $E_L, E_R \subseteq \Gamma^*$ are regular languages called *left* and *right constraints*.
  - *Accepting* meta-instruction $(E, Accept)$, where $E \subseteq \Gamma^*$ is a regular language.

Language of Restarting Automaton

- Rewriting meta-instructions of M induce a *reducing relation* $\vdash_M \subseteq \Gamma^* x \Gamma^*$ such that:

  for each $u, v \in \Gamma^*$, $u \vdash_M v$ if and only if there exist an instruction $i = (E_L, x \rightarrow y, E_R)$ in I and words $u_1, u_2 \in \Gamma^*$ such that $u = u_1 x u_2$, $v = u_1 y u_2$, $u_1 \in E_L$ and $u_2 \in E_R$.

Language of Restarting Automaton

- Rewriting meta-instructions of M induce a *reducing relation* $\vdash_M \subseteq \Gamma^* x \Gamma^*$ such that:

  for each $u, v \in \Gamma^*$, $u \vdash_M v$ if and only if there exist an instruction $i = (E_L, x \rightarrow y, E_R)$ in I and words $u_1, u_2 \in \Gamma^*$ such that $u = u_1 x u_2$, $v = u_1 y u_2$, $u_1 \in E_L$ and $u_2 \in E_R$.

- Accepting meta-instructions of M define *simple sentential forms* S_M = set of words $u \in \Gamma^*$, for which there exist an instruction $i = (E, Accept)$ in I such that $u \in E$.

Language of Restarting Automaton

- Rewriting meta-instructions of M induce a *reducing relation* $\vdash_M \subseteq \Gamma^* x \Gamma^*$ such that:

  for each $u, v \in \Gamma^*$, $u \vdash_M v$ if and only if there exist an instruction $i = (E_L, x \rightarrow y, E_R)$ in I and words $u_1, u_2 \in \Gamma^*$ such that $u = u_1 x u_2$, $v = u_1 y u_2$, $u_1 \in E_L$ and $u_2 \in E_R$.
- Accepting meta-instructions of M define *simple sentential forms* S_M = set of words $u \in \Gamma^*$, for which there exist an instruction $i = (E, Accept)$ in I such that $u \in E$.
- The *input language* of M is defined as:

  $L(M) = \{u \in \Sigma^* / \exists v \in S_M : u \vdash_M^* v\}$,

  where $\vdash_M^*$ is a reflexive and transitive closure of $\vdash_M$.

Example

- How to create a *restarting automaton* that recognizes the language $L = \{a^i b^i c^j d^j \mid i, j > 0\}$.
- *Accepting meta-instructions:*

Name	Accepting Language
A0	^abcd$

- *Reducing (or rewriting) meta-instructions:*

Name	Left Language	From Word	To Word	Right Language
R0	^a*$	ab	λ	^b*c*d*$
R1	^a*b*c*$	cd	λ	^d*$

Example

- Suppose that we have a word: *aaabbbccdd*

Name	Accepting Lang.
A0	^abcd$

Name	Left Lang.	From Word	To Word	Right Lang.
R0	^a*$	ab	λ	^b*c*d*$
R1	^a*b*c*$	cd	λ	^d*$

Example

- $aa\underline{ab}bbccdd \xrightarrow{R0} aabbccdd$

Name	Accepting Lang.
A0	^abcd$

Name	Left Lang.	From Word	To Word	Right Lang.
R0	^a*$	ab	λ	^b*c*d*$
R1	^a*b*c*$	cd	λ	^d*$

Example

- $aaabbbccdd \xrightarrow{R0} aabbccdd \xrightarrow{R1} aabbcd$

Name	Accepting Lang.
A0	^abcd$

Name	Left Lang.	From Word	To Word	Right Lang.
R0	^a*$	ab	λ	^b*c*d*$
R1	^a*b*c*$	cd	λ	^d*$

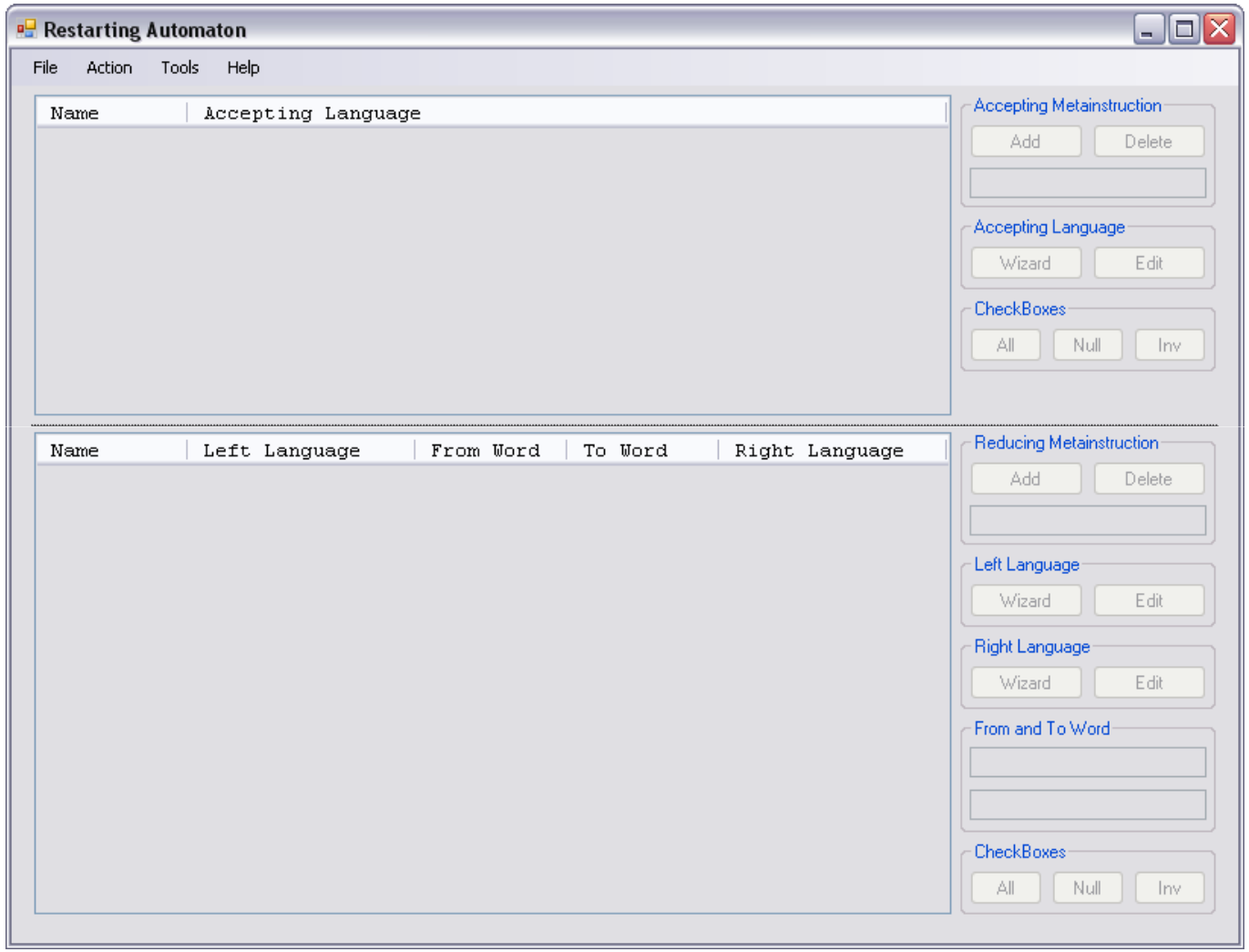
Example

- *aaabbbccdd* $\overset{R0}{\rightarrow}$ *aabbccdd* $\overset{R1}{\rightarrow}$ *aabbcd* $\overset{R0}{\rightarrow}$ *abcd*
- *abcd* is accepted by *A0*, so the whole word *aaabbbccdd* is accepted.
- Note that *abcd* is a simple sentential form.

Name	Accepting Lang.
A0	^abcd$

Name	Left Lang.	From Word	To Word	Right Lang.
R0	^a*$	ab	λ	^b*c*d*$
R1	^a*b*c*$	cd	λ	^d*$

PART I: RestartingAutomaton.exe



Capabilities and features

- **Design** a restarting automaton. The design of restarting automaton consists of stepwise design of accepting and reducing meta-instructions. You can save (load) restarting automaton to (from) an XML file.
- **Test** correctly defined restarting automaton:
  - The system is able to give you a list of all words that can be obtained by reductions from a given word w.
  - The system is able to give you a list of all reduction paths from one given word to another given word.
- Start a **server mode**, in which the client applications can use services provided by the server application.
- You can use **specialized tools** to define formal languages.
- You can also save, load, copy, paste and view an **XML** representation of the actual state of every tool.

Learning Languages

- There are several tools that are used to define languages:
  - **DFA Modeler**: allows you to enter a regular language by specifying its underlying *deterministic finite automaton*.
  - **LStar Algorithm**: encapsulates *Dana Angluin's L* algorithm* that is a machine learning algorithm which learns deterministic finite automaton using *membership* and *equivalence queries*.
  - **RPNI Algorithm**: encapsulates a machine learning algorithm which learns deterministic finite automaton based on a given set of *labeled examples*.
  - **Regular Expression**: allows you to enter a regular language by specifying the *regular expression*.
  - **SLT Language**: allows you to design a regular language by specifying a positive integer *k* and *positive examples* using the algorithm for learning *k-SLT languages*.

Pros and Cons

- **Pros**:
  - The application is written in C# using .NET Framework 2.0. It works both on Win32 and UNIX platforms.
  - The application demonstrates that it is easy to design and work with restarting automata.
  - Any component of the application can be easily reused in another projects. It safes your work.
- **Cons**:
  - The application is a toy that allows you only to design simple restarting automata recognizing only simple formal languages with small alphabets consisting of few letters. On large inputs the computation can take a long time and it can produce a huge output.

PART II: RA_{CL}

- k-local Restarting Automaton CLEARING (k-RA_{CL}) $M = (\Sigma, I)$
  - Σ is a finite nonempty alphabet, $¢, \$ \notin \Sigma$
  - I is a finite set of instructions (x, z, y), $x \in LC_k$, $y \in RC_k$, $z \in \Sigma^+$
    - left context $LC_k = \Sigma^k \cup ¢.\Sigma^{\leq k-1}$
    - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$

PART II: RA_{CL}

- k-local Restarting Automaton CLEARING (k-RA_{CL}) $M = (\Sigma, I)$
  - Σ is a finite nonempty alphabet, $¢, \$ \notin \Sigma$
  - I is a finite set of instructions (x, z, y), $x \in LC_k$, $y \in RC_k$, $z \in \Sigma^+$
    - left context $LC_k = \Sigma^k \cup ¢.\Sigma^{\leq k-1}$
    - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$
- A word $w = uzv$ can be rewritten to uv ($uzv \rightarrow uv$) if and only if there exist an instruction $i = (x, z, y) \in I$ such that:
  - $x \sqsupseteq ¢.u$ (x is a suffix of $¢.u$)
  - $y \sqsubseteq v.\$$ (y is a prefix of $v.\$$)

PART II: RA_{CL}

- k-local Restarting Automaton CLEARING (k-RA_{CL}) $M = (\Sigma, I)$
  - Σ is a finite nonempty alphabet, $¢, \$ \notin \Sigma$
  - I is a finite set of instructions (x, z, y), $x \in LC_k$, $y \in RC_k$, $z \in \Sigma^+$
    - left context $LC_k = \Sigma^k \cup ¢.\Sigma^{\leq k-1}$
    - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$
- A word $w = uzv$ can be rewritten to uv ($uzv \rightarrow uv$) if and only if there exist an instruction $i = (x, z, y) \in I$ such that:
  - $x \sqsupseteq ¢.u$ (x is a suffix of $¢.u$)
  - $y \sqsubseteq v.\$$ (y is a prefix of $v.\$$)
- A word w is accepted if and only if $w \rightarrow^* \lambda$ where $\rightarrow^*$ is reflexive and transitive closure of $\rightarrow$.

PART II: RA_{CL}

- k-local Restarting Automaton CLEARING (k-RA_{CL}) $M = (\Sigma, I)$
  - Σ is a finite nonempty alphabet, $¢, \$ \notin \Sigma$
  - I is a finite set of instructions (x, z, y), $x \in LC_k$, $y \in RC_k$, $z \in \Sigma^+$
    - left context $LC_k = \Sigma^k \cup ¢.\Sigma^{\leq k-1}$
    - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$
- A word $w = uzv$ can be rewritten to uv ($uzv \rightarrow uv$) if and only if there exist an instruction $i = (x, z, y) \in I$ such that:
  - $x \sqsupseteq ¢.u$ (x is a suffix of $¢.u$)
  - $y \sqsubseteq v.\$$ (y is a prefix of $v.\$$)
- A word w is accepted if and only if $w \rightarrow^* \lambda$ where $\rightarrow^*$ is reflexive and transitive closure of $\rightarrow$.
- We define the class RA_{CL} as $\bigcup_{k \geq 1} k\text{-}RA_{CL}$.

Why RA_{CL}?

- This model was inspired by the *Associative Language Descriptions* (*ALD*) model
  - By Alessandra Cherubini, Stefano Crespi-Reghizzi, Matteo Pradella, Pierluigi San Pietro
  - See: http://home.dei.polimi.it/sanpietr/ALD/ALD.html
- The more restricted model we have the easier is the investigation of its properties. Moreover, the learning methods are much more simple and straightforward.

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$
- For instance:
  - $aaaabbbb \xrightarrow{R1} aaabbb$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$
- For instance:
  - $aaaabbbb \vdash^{R1} aaabbb \vdash^{R1} aabb$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$
- For instance:
  - $aaaabbbb \vdash^{R1} aaabbb \vdash^{R1} aabb \vdash^{R1} ab$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$
- For instance:
  - $aaaabbbb \vdash^{R1} aaabbb \vdash^{R1} aabb \vdash^{R1} \underline{ab} \vdash^{R2} \lambda$
- Now we see that the word *aaaabbbb* is accepted because $aaaabbbb \rightarrow^* \lambda$.

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- $1\text{-}RA_{CL}$ $M = (\{a, b\}, I)$ where the instructions I are:
  - $R1 = (a, ab, b)$
  - $R2 = (¢, ab, \$)$
- For instance:
  - $aaaabbbb \rightarrow_{R1} aaabbb \rightarrow_{R1} aabb \rightarrow_{R1} \underline{ab} \rightarrow_{R2} \lambda$
- Now we see that the word *aaaabbbb* is accepted because $aaaabbbb \rightarrow^* \lambda$.
- Note that λ is always accepted because $\lambda \rightarrow^* \lambda$.

Some Theorems

- **Theorem**: For every finite $L \subseteq \Sigma^*$ there exist $1\text{-}RA_{CL}$ M such that $L(M) = L$.
  - **Proof**. Suppose $L = \{w_1, ..., w_n\}$.
    Consider $I = \{(¢, w_1, \$), ..., (¢, w_n, \$)\}$. ■

Some Theorems

- **Theorem**: For every finite $L \subseteq \Sigma^*$ there exist $1\text{-}RA_{CL}$ M such that $L(M) = L$.
  - **Proof**. Suppose $L = \{w_1, ..., w_n\}$.

    Consider $I = \{(¢, w_1, \$), ..., (¢, w_n, \$)\}$. ■
- **Theorem**: For all $k \geq 1$ $\mathcal{L}(k\text{-}RA_{CL}) \subseteq \mathcal{L}((k+1)\text{-}RA_{CL})$.

Some Theorems

- **Theorem**: For every finite $L \subseteq \Sigma^*$ there exist $1\text{-}RA_{CL}$ M such that $L(M) = L$.
  - **Proof**. Suppose $L = \{w_1, ..., w_n\}$.

    Consider $I = \{(¢, w_1, \$), ..., (¢, w_n, \$)\}$. ■
- **Theorem**: For all $k \geq 1$ $\mathcal{L}(k\text{-}RA_{CL}) \subseteq \mathcal{L}((k+1)\text{-}RA_{CL})$.
- **Theorem**: For each regular language L there exist a k-RA_{CL} M such that $L(M) = L \cup \{\lambda\}$.

Some Theorems

- **Theorem**: For every finite $L \subseteq \Sigma^*$ there exist $1\text{-}RA_{CL}$ M such that $L(M) = L$.
  - **Proof**. Suppose $L = \{w_1, ..., w_n\}$.
    Consider $I = \{(¢, w_1, \$), ..., (¢, w_n, \$)\}$. ■
- **Theorem**: For all $k \geq 1$ $\mathcal{L}(k\text{-}RA_{CL}) \subseteq \mathcal{L}((k+1)\text{-}RA_{CL})$.
- **Theorem**: For each regular language L there exist a $k\text{-}RA_{CL}$ M such that $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Based on *pumping lemma* for *regular languages*.
  - For each $z \in \Sigma^*$, $|z|=n$ there exist u, v, w such that $|v| \geq 1$, $\delta(q_0, uv) = \delta(q_0, u)$; the word v can be crossed out.
  - We add corresponding instruction $i_z = (¢u, v, w)$.
  - For each accepted $z \in \Sigma^{<n}$ we add instruction $i_z = (¢, z, \$)$. ■

Some Theorems

- **Lemma**: Let M be RA_{CL}, $i = (x, z, y)$ its instruction and $w = uv$ such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.

  Then $uv \in L(M) \Rightarrow uzv \in L(M)$.

  - **Proof**. $uzv \multimap^{i} uv \to^{*} \lambda$. ■

Some Theorems

- **Lemma**: Let M be RA_{CL}, $i = (x, z, y)$ its instruction and $w = uv$ such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.

  Then $uv \in L(M) \Rightarrow uzv \in L(M)$.

  - **Proof**. $uzv \rightarrow^{i} uv \rightarrow^{*} \lambda$. ■

- **Theorem**: Languages $L_1 \cup L_2$ and $L_1.L_2$

  - where $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \{a^n b^{2n} \mid n \geq 0\}$

  are not accepted by any RA_{CL}.

  - **Proof** by contradiction, based on the previous lemma.

Some Theorems

- **Lemma**: Let M be RA_{CL}, $i = (x, z, y)$ its instruction and $w = uv$ such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.

  Then $uv \in L(M) \Rightarrow uzv \in L(M)$.

  - **Proof**. $uzv \xrightarrow{i} uv \rightarrow^* \lambda$. ■
- **Theorem**: Languages $L_1 \cup L_2$ and $L_1.L_2$
  - where $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \{a^n b^{2n} \mid n \geq 0\}$

  are not accepted by any RA_{CL}.
  - **Proof** by contradiction, based on the previous lemma.
- **Corollary**: RA_{CL} is not closed under *union* and *concatenation*.

Some Theorems

- **Lemma**: Let M be RA_{CL}, $i = (x, z, y)$ its instruction and $w = uv$ such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.

  Then $uv \in L(M) \Rightarrow uzv \in L(M)$.

  - **Proof**. $uzv \xrightarrow{i} uv \rightarrow^* \lambda$. ■
- **Theorem**: Languages $L_1 \cup L_2$ and $L_1.L_2$
  - where $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \{a^n b^{2n} \mid n \geq 0\}$

  are not accepted by any RA_{CL}.

  - **Proof** by contradiction, based on the previous lemma.
- **Corollary**: RA_{CL} is not closed under *union* and *concatenation*.
- **Corollary**: RA_{CL} is not closed under *homomorphism*.
  - **Consider** $\{a^n b^n \mid n \geq 0\} \cup \{c^n d^{2n} \mid n \geq 0\}$ and homomorphism defined as: $a \mapsto a$, $b \mapsto b$, $c \mapsto a$, $d \mapsto b$. ■

Some Theorems

- **Theorem**: The language $L_1 = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$ is not accepted by any RA_{CL}.
- **Theorem**: The languages:
  - $L_2 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_3 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_4 = \{a^m b^m \mid m \geq 0\}$

  are recognized by $1\text{-}RA_{CL}$.

Some Theorems

- **Theorem**: The language $L_1 = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$ is not accepted by any RA_{CL}.
- **Theorem**: The languages:
  - $L_2 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_3 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_4 = \{a^m b^m \mid m \geq 0\}$

  are recognized by $1\text{-}RA_{CL}$.
- **Corollary**: RA_{CL} is not closed under intersection.
  - **Proof**. $L_1 = L_2 \cap L_3$. ■

Some Theorems

- **Theorem**: The language $L_1 = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$ is not accepted by any RA_{CL}.
- **Theorem**: The languages:
  - $L_2 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_3 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_4 = \{a^m b^m \mid m \geq 0\}$

  are recognized by $1\text{-}RA_{CL}$.
- **Corollary**: RA_{CL} is not closed under intersection.
  - **Proof**. $L_1 = L_2 \cap L_3$. ■
- **Corollary**: RA_{CL} is not closed under intersection with a regular language.
  - **Proof**. L_3 is a *regular language*. ■

Some Theorems

- **Theorem**: The language $L_1 = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$ is not accepted by any RA_{CL}.
- **Theorem**: The languages:
  - $L_2 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_3 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_4 = \{a^m b^m \mid m \geq 0\}$

  are recognized by $1\text{-}RA_{CL}$.
- **Corollary**: RA_{CL} is not closed under intersection.
  - **Proof**. $L_1 = L_2 \cap L_3$. ■
- **Corollary**: RA_{CL} is not closed under intersection with a regular language.
  - **Proof**. L_3 is a *regular language*. ■
- **Corollary**: RA_{CL} is not closed under difference.
  - **Proof**. $L_1 = (L_2 - L_4) \cup \{\lambda\}$. ■

Parentheses

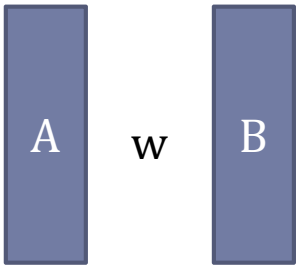
- The following instruction of $1\text{-}RA_{CL}$ M is enough for recognizing the language of correct parentheses:
  - $(\lambda, (\,), \lambda)$

Parentheses

- The following instruction of $1\text{-}RA_{CL}$ M is enough for recognizing the language of correct parentheses:
  - $(\lambda, (\ ), \lambda)$
- **Note**: This instruction represents a *set of instructions*:
  - $(\{¢\} \cup \Sigma, (\ ), \Sigma \cup \{\$\})$, where $\Sigma = \{(,)\}$ and
  - $(A, w, B) = \{(a, w, b) \mid a \in A, b \in B\}$.

Parentheses

- The following instruction of $1\text{-}RA_{CL}$ M is enough for recognizing the language of correct parentheses:
  - $(\lambda, (\,), \lambda)$
- **Note**: This instruction represents a *set of instructions*:
  - $(\{¢\} \cup \Sigma, (\,), \Sigma \cup \{\$\})$, where $\Sigma = \{(,)\}$ and
  - $(A, w, B) = \{(a, w, b) \mid a \in A, b \in B\}$.
- **Note**: We use the following notation for the (A, w, B):

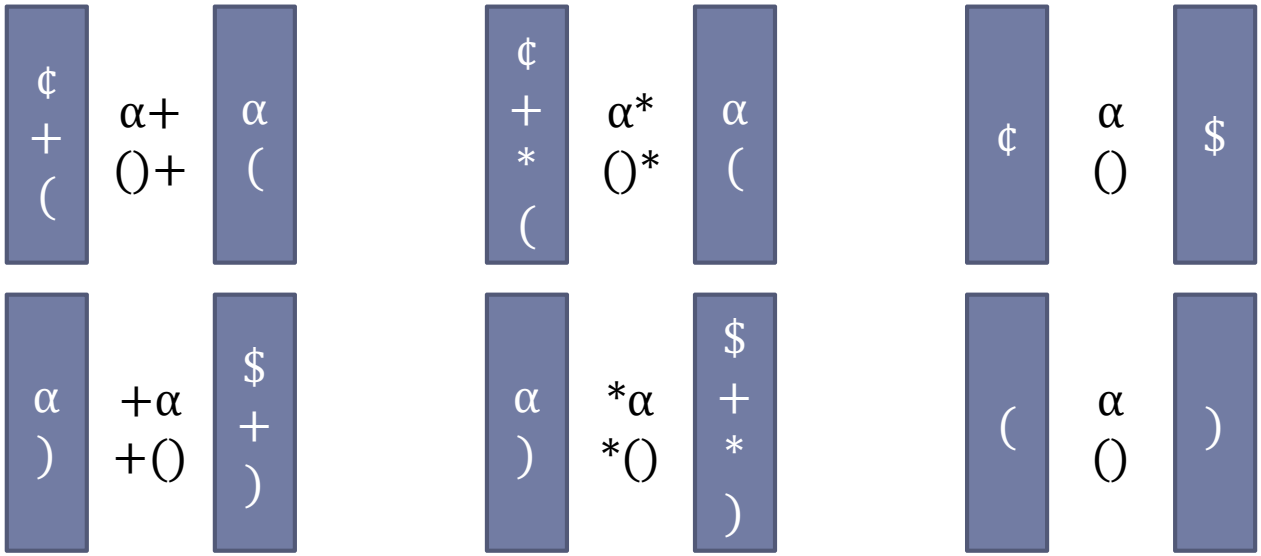


Arithmetic expressions

- Suppose that we want to check correctness of arithmetic expressions over the alphabet $\Sigma = \{\alpha, +, *, (,)\}$.
  - For example $\alpha+(\alpha*\alpha+\alpha)$ is correct, $\alpha*+\alpha$ is not.
  - The priority of the operations is considered.

Arithmetic expressions

- Suppose that we want to check correctness of arithmetic expressions over the alphabet $\Sigma = \{\alpha, +, *, (,)\}$.
  - For example $\alpha+(\alpha*\alpha+\alpha)$ is correct, $\alpha*+\alpha$ is not.
  - The priority of the operations is considered.
- The following *1-RA*$_{CL}$ *M* is sufficient:



Arithmetic expressions: Example

Expression	Instruction
$\underline{\alpha^*}\alpha + ((\alpha + \alpha) + (\alpha + \alpha^*\alpha))^*\alpha$	(¢, α*, α)
$\alpha + ((\alpha \underline{+ \alpha}) + (\alpha + \alpha^*\alpha))^*\alpha$	(α, +α,))
$\alpha + ((\alpha) + (\alpha + \alpha^*\alpha))\underline{^*\alpha}$	( ), *α, $)
$\alpha + ((\alpha) + (\alpha + \underline{\alpha^*}\alpha))$	(+, α*, α)
$\alpha + ((\alpha) + (\underline{\alpha +} \alpha))$	((, α+, α)
$\alpha + ((\underline{\alpha}) + (\alpha))$	((, α,))
$\alpha + (\underline{( ) +} (\alpha))$	((, ( )+, ( )
$\alpha + ((\underline{\alpha}))$	((, α,))
$\alpha + (\underline{( )})$	((, ( ),))
$\underline{\alpha +} ( )$	(¢, α+, ( )
$\underline{( )}$	(¢, ( ), $)
λ	accept

Nondeterminism

- Assume the following instructions:
  - *R1 = (bb, a, bbbb)*
  - *R2 = (bb, bb, $)*
  - *R3 = (¢, cbb, $)*

  and the word: *cbbabbbb*.

Nondeterminism

- Assume the following instructions:
  - *R1 = (bb, a, bbbb)*
  - *R2 = (bb, bb, $)*
  - *R3 = (¢, cbb, $)*

  and the word: *cbbabbbb*. Then:
  - $cbb\underline{a}bbbb \xrightarrow{R1} cbbbb\underline{bb} \xrightarrow{R2} cbb\underline{bb} \xrightarrow{R2} \underline{cbb} \xrightarrow{R3} \lambda$.

Nondeterminism

- Assume the following instructions:
  - *R1 = (bb, a, bbbb)*
  - *R2 = (bb, bb, $)*
  - *R3 = (¢, cbb, $)*

  and the word: *cbbabbbb*. Then:
  - $cbbabbbb \xrightarrow{R1} cbbbbbb \xrightarrow{R2} cbbbb \xrightarrow{R2} cbb \xrightarrow{R3} \lambda$.
- **But** if we have started with *R2*:
  - $cbbabbbb \xrightarrow{R2} cbbabb$

  then it would not be possible to continue.

Nondeterminism

- Assume the following instructions:
  - *R1 = (bb, a, bbbb)*
  - *R2 = (bb, bb, $)*
  - *R3 = (¢, cbb, $)*

  and the word: *cbbabbbb*. Then:
  - $cbbabbbb \xrightarrow{R1} cbbbbbb \xrightarrow{R2} cbbbb \xrightarrow{R2} cbb \xrightarrow{R3} \lambda$.
- **But** if we have started with *R2*:
  - $cbbabbbb \xrightarrow{R2} cbbabb$

  then it would not be possible to continue.
- **⇒ The order of used instructions is important!**

Hardest CFL H

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*

Hardest CFL H

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let D_2 be *Semi-Dyck* set on $\{a_1, a_2, a_1', a_2'\}$ generated by the grammar: $S \rightarrow a_1 S a_1' S \mid a_2 S a_2' S \mid \lambda$.

Hardest CFL H

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let D_2 be *Semi-Dyck* set on $\{a_1, a_2, a_1', a_2'\}$ generated by the grammar: $S \rightarrow a_1 S a_1' S \mid a_2 S a_2' S \mid \lambda$.

  Let $\Sigma = \{a_1, a_2, a_1', a_2', b, c\}$, $d \notin \Sigma$.

Hardest CFL H

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let D_2 be *Semi-Dyck* set on $\{a_1, a_2, a_1', a_2'\}$ generated by the grammar: $S \rightarrow a_1 S a_1' S \mid a_2 S a_2' S \mid \lambda$.

  Let $\Sigma = \{a_1, a_2, a_1', a_2', b, c\}$, $d \notin \Sigma$.

  Then $H = \{\lambda\} \cup \{\prod_{i=1..n} x_i c y_i c z_i d \mid n \geq 1, y_1 y_2 \ldots y_n \in b D_2, x_i, z_i \in \Sigma^*\}$.

Hardest CFL H

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let D_2 be *Semi-Dyck* set on $\{a_1, a_2, a_1', a_2'\}$ generated by the grammar: $S \rightarrow a_1 S a_1' S \mid a_2 S a_2' S \mid \lambda$.

  Let $\Sigma = \{a_1, a_2, a_1', a_2', b, c\}$, $d \notin \Sigma$.

  Then $H = \{\lambda\} \cup \{\prod_{i=1..n} x_i c y_i c z_i d \mid n \geq 1, y_1 y_2 \ldots y_n \in b D_2, x_i, z_i \in \Sigma^*\}$.
- **Each *CFL* can be represented as an inverse homomorphism of *H*.**

Hardest CFL H

- **Theorem**: H is not accepted by any RA_{CL}.
  - **Proof** by contradiction.
- **But** if we slightly extend the definition of RA_{CL} then we will be able to recognize H.

$RA_{\Delta CL}$

- k-local Restarting Automaton $_{\Delta CLEARING}$

  $k\text{-}RA_{\Delta CL}\ M = (\Sigma, I)$

  - Σ is a finite nonempty alphabet, $¢, \$, \Delta \notin \Sigma, \Gamma = \Sigma \cup \{\Delta\}$
  - I is a finite set of instructions:
    - (1) $(x, z \rightarrow \lambda, y)$
    - (2) $(x, z \rightarrow \Delta, y)$
  - where $x \in LC_k$, $y \in RC_k$, $z \in \Gamma^+$.
    - left context $LC_k = \Gamma^k \cup ¢.\ \Gamma^{\leq k-1}$
    - right context $RC_k = \Gamma^k \cup \Gamma^{\leq k-1}.\$$

$RA_{\Delta CL}$

- A word $w = uzv$ can be rewritten to uv ($uzv \rightarrow uv$) if and only if there exist an instruction $i = (x, z \rightarrow \lambda, y) \in I$

$RA_{\Delta CL}$

- A word $w = uzv$ can be rewritten to uv ($uzv \rightarrow uv$) if and only if there exist an instruction $i = (x, z \rightarrow \lambda, y) \in I$
- … or to $u\Delta v$ ($uzv \rightarrow u\Delta v$) if and only if there exist an instruction $i = (x, z \rightarrow \Delta, y) \in I$

$\mathrm{RA}_{\Delta\mathrm{CL}}$

- A word $w = uzv$ can be rewritten to uv ($uzv \to uv$) if and only if there exist an instruction $i = (x, z \to \lambda, y) \in I$
- … or to $u\Delta v$ ($uzv \to u\Delta v$) if and only if there exist an instruction $i = (x, z \to \Delta, y) \in I$
- such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$.$

$RA_{\Delta CL}$

- A word $w = uzv$ can be rewritten to uv ($uzv \to uv$) if and only if there exist an instruction $i = (x, z \to \lambda, y) \in I$
- … or to $u\Delta v$ ($uzv \to u\Delta v$) if and only if there exist an instruction $i = (x, z \to \Delta, y) \in I$
- such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.
- A word w is accepted if and only if $w \to^* \lambda$ where $\to^*$ is reflexive and transitive closure of $\to$.

$RA_{\Delta CL}$

- A word $w = uzv$ can be rewritten to uv ($uzv \to uv$) if and only if there exist an instruction $i = (x, z \to \lambda, y) \in I$
- … or to $u\Delta v$ ($uzv \to u\Delta v$) if and only if there exist an instruction $i = (x, z \to \Delta, y) \in I$
- such that $x \sqsupseteq ¢.u$ and $y \sqsubseteq v.\$$.
- A word w is accepted if and only if $w \to^* \lambda$ where $\to^*$ is reflexive and transitive closure of $\to$.
- We define the class $RA_{\Delta CL}$ as $\bigcup_{k \geq 1} k\text{-}RA_{\Delta CL}$.

Hardest CFL H revival

- **Theorem**: H is recognized by $1\text{-}RA_{\Delta CL}$.
  - **Idea**. Suppose that we have $w \in H$:

    $w = ¢\, x_1 c y_1 c z_1 d x_2 c y_2 c z_2 d \ldots x_n c y_n c z_n d\, \$$
  - In the *first phase* we start with deleting letters (from the alphabet $\Sigma = \{a_1, a_2, a_1', a_2', b, c\}$) from the right side of ¢ and from the left and right sides of the letters d.
  - As soon as we think that we have the following word:

    $¢\, c y_1 c d c y_2 c d \ldots c y_n c d\, \$$

    we introduce the Δ symbols:

    $¢\, \Delta y_1 \Delta y_2 \Delta \ldots \Delta y_n \Delta\, \$$
  - In the *second phase* we check if $y_1 y_2 \ldots y_n \in bD_2$.

Instructions of M recognizing CFL H

- Suppose $\Sigma = \{a_1, a_2, a_1', a_2', b, c\}$, $d \notin \Sigma$, $\Gamma = \Sigma \cup \{d, \Delta\}$.

Instructions for the first phase:	Instructions for the second phase:
(1) $(¢, \Sigma \to \lambda, \Sigma)$	(7) $(\Gamma, a_1a_1' \to \lambda, \Gamma - \{b\})$
(2) $(\Sigma, \Sigma \to \lambda, d)$	(8) $(\Gamma, a_2a_2' \to \lambda, \Gamma - \{b\})$
(3) $(d, \Sigma \to \lambda, \Sigma)$	(9) $(\Gamma, a_1\Delta a_1' \to \Delta, \Gamma - \{b\})$
(4) $(¢, c \to \Delta, \Sigma \cup \{\Delta\})$	(10) $(\Gamma, a_2\Delta a_2' \to \Delta, \Gamma - \{b\})$
(5) $(\Sigma \cup \{\Delta\}, cdc \to \Delta, \Sigma \cup \{\Delta\})$	(11) $(\Sigma - \{c\}, \Delta \to \lambda, \Delta)$
(6) $(\Sigma \cup \{\Delta\}, cd \to \Delta, \$)$	(12) $(¢, \Delta b\Delta \to \lambda, \$)$

- In fact, there is no such thing as a first phase or a second phase. We have only instructions.
- **Theorem**: $H \subseteq L(M)$.
- **Theorem**: $H \supseteq L(M)$.
  - **Idea**. We describe all words that are generated by the instructions.

The power of RA_{CL}

- **Theorem**: There exists a $k\text{-}RA_{CL}$ M recognizing a language that is **not a *CFL***.

The power of RA_{CL}

- **Theorem**: There exists a $k\text{-}RA_{CL}$ M recognizing a language that is **not a *CFL***.
  - **Idea**. We try to create a $k\text{-}RA_{CL}$ M such that $L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2m} \mid m \geq 0\}$.

The power of RA_{CL}

- **Theorem**: There exists a $k\text{-}RA_{CL}$ M recognizing a language that is **not a *CFL***.
  - **Idea**. We try to create a $k\text{-}RA_{CL}$ M such that $L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2m} \mid m \geq 0\}$.
  - If $L(M)$ is a *CFL* then the intersection with a regular language is also a *CFL*. In our case the intersection is not a *CFL*.

How does it work

- **Example:**

  ¢ abababababababab $

How does it work

- **Example:**

  ¢ *abababababababab* $ → ¢ *abababababababb* $ →

  ¢ *abababababbabb* $ → ¢ *ababababbabbabb* $ →

  ¢ *abbabbabbabb* $

How does it work

- **Example:**

  ¢ *abababababababab* $ → ¢ *abababababababb* $ →

  ¢ *abababababbabb* $ → ¢ *ababababbabbabb* $ →

  ¢ *abbabbabbabb* $ → ¢ *abbabbabbbab* $ →

  ¢ *abbabbabab* $ → ¢ *abbababab* $ →

  ¢ *abababab* $

How does it work

- Example:

  ¢ *abababababababab* $ → ¢ *abababababababb* $ →

  ¢ *abababababbabb* $ → ¢ *abababbabbabb* $ →

  ¢ *abbabbabbabb* $ → ¢ *abbabbabbab* $ →

  ¢ *abbabbabab* $ → ¢ *abbababab* $ →

  ¢ *abababab* $ → ¢ *abababb* $ →

  ¢ *abbabb* $

How does it work

- **Example:**

  ¢ *abababababababab* $ → ¢ *abababababababb* $ →

  ¢ *abababababbabb* $ → ¢ *abababbabbabb* $ →

  ¢ *abbabbabbabb* $ → ¢ *abbabbabbab* $ →

  ¢ *abbabbabab* $ → ¢ *abbababab* $ →

  ¢ *abababab* $ → ¢ *ababbb* $ →

  ¢ *abbabb* $ → ¢ *abbab* $ →

  ¢ *abab* $

How does it work

- **Example:**

  ¢ *abababababababab* $ → ¢ *abababababababb* $ →

  ¢ *abababababbabb* $ → ¢ *abababbabbabb* $ →

  ¢ *abbabbabbabb* $ → ¢ *abbabbabbab* $ →

  ¢ *abbabbabab* $ → ¢ *abbababab* $ →

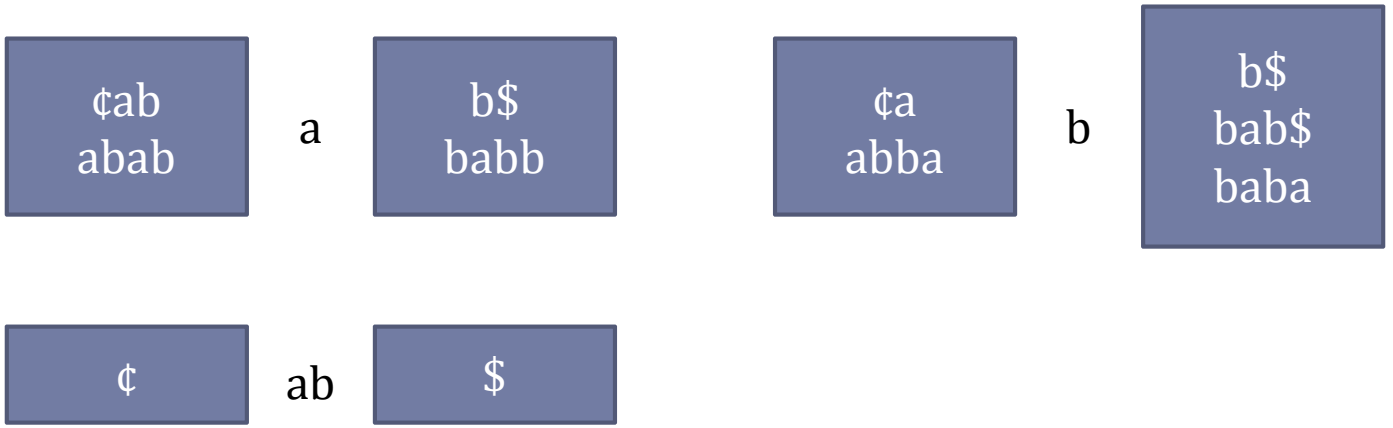
  ¢ *abababab* $ → ¢ *ababbb* $ →

  ¢ *abbabb* $ → ¢ *abbab* $ →

  ¢ *abab* $ → ¢ *abb* $ → ¢ *ab* $ → ¢ λ $ → ***accept.***

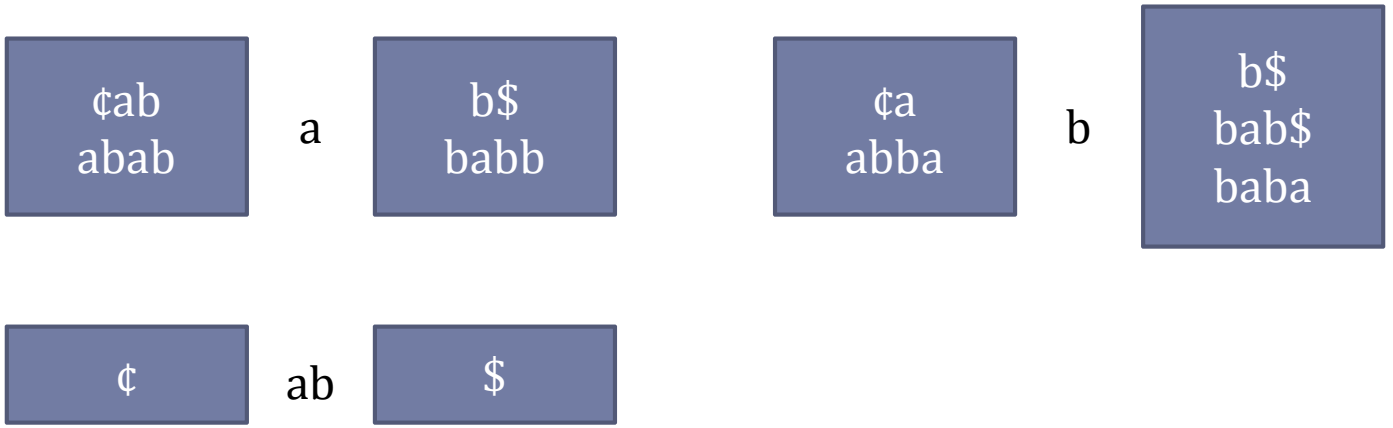
The power of RA_{CL}: Instructions

- If we infer instructions from the previous example, then for $k=4$ we get the following $4\text{-}RA_{CL}$ M:



The power of RA_{CL}: Instructions

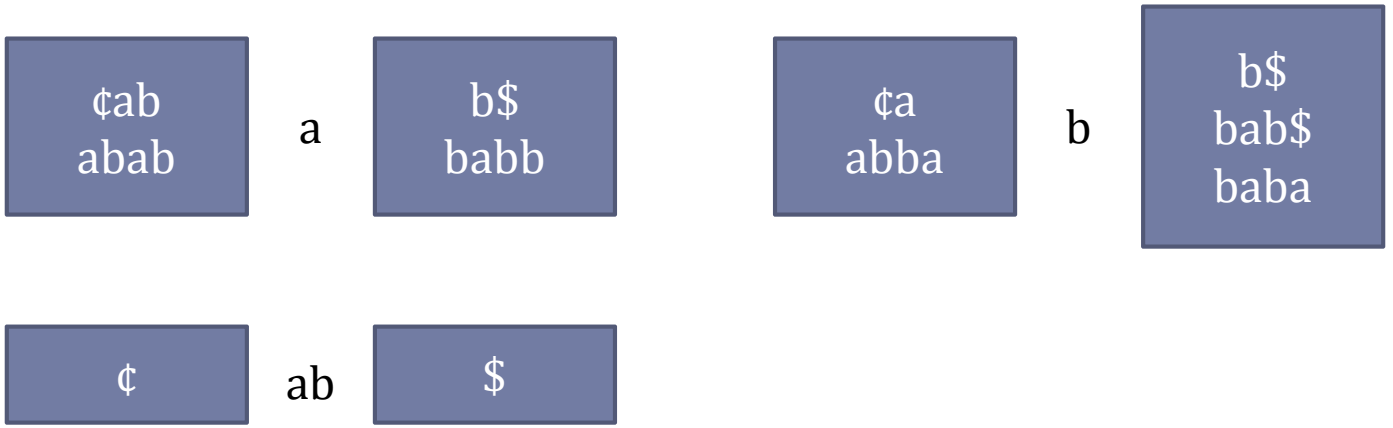
- If we infer instructions from the previous example, then for $k=4$ we get the following $4\text{-}RA_{CL}$ M:



- **Theorem**: $L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2m} \mid m \geq 0\}$.
  - **Idea**. We describe the whole language $L(M)$.

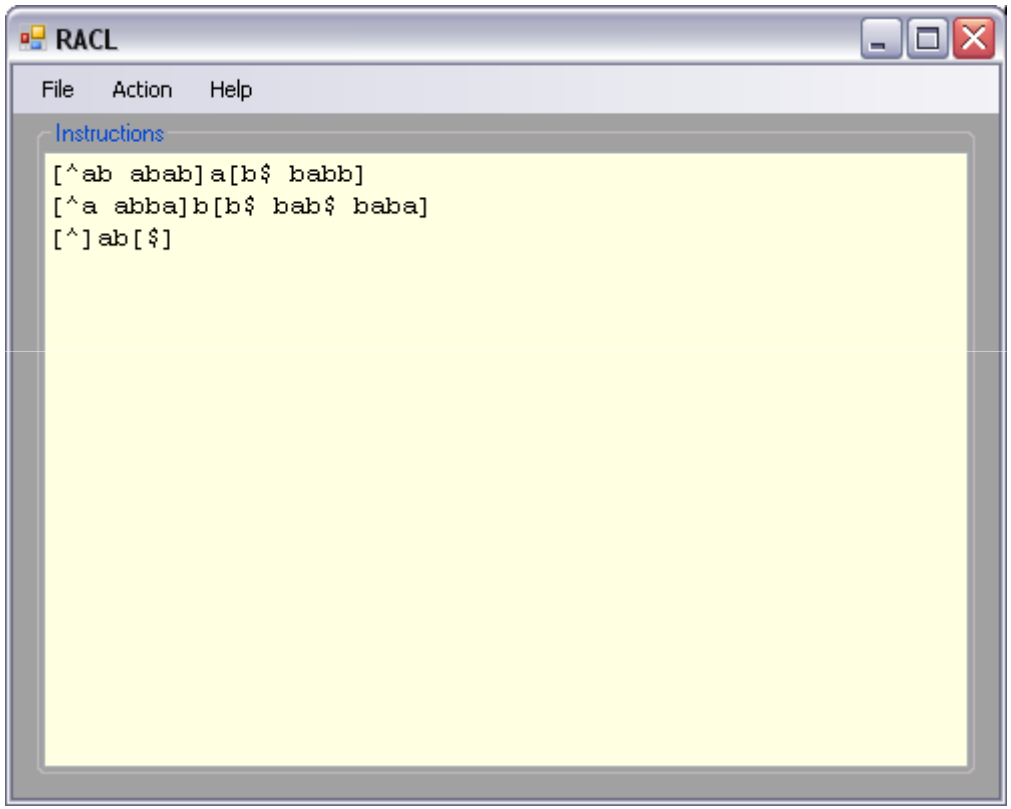
The power of RA_{CL}: Instructions

- If we infer instructions from the previous example, then for $k=4$ we get the following $4\text{-}RA_{CL}$ M:

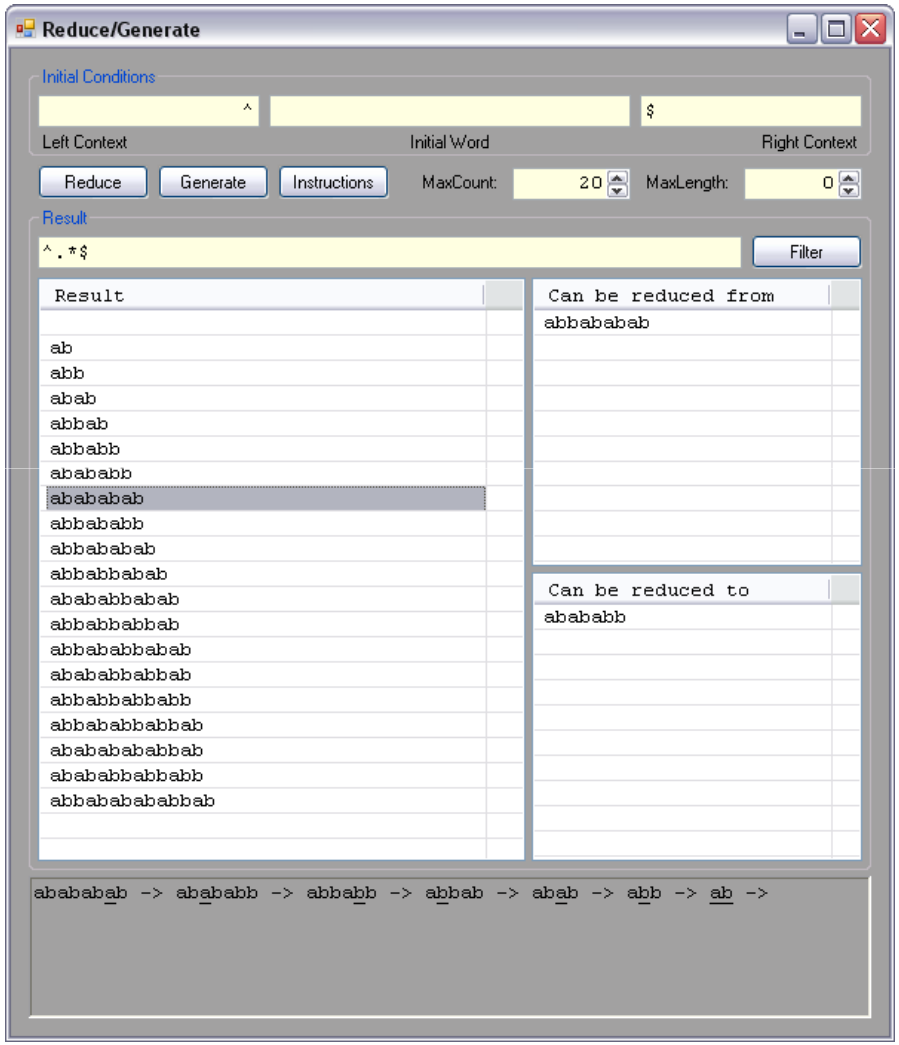


- **Theorem**: $L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2m} \mid m \geq 0\}$.
  - **Idea**. We describe the whole language $L(M)$.
- Note that this technique does not work with $k<4$.

PART III: RACL.exe



RACL.exe: Reduce/Generate



PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols ¢ and $. I think that this model is useless because *it is not able to recognize even finite languages*.

PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols $¢$ and $\$$. I think that this model is useless because *it is not able to recognize even finite languages*.
- We can generalize our model to a $k\text{-}RA_{\Delta nCL}$ which is the same model as the $k\text{-}RA_{\Delta CL}$ except that it uses n Δ symbols: Δ_1, Δ_2, ..., Δ_n. This model *is able to recognize each CFL*.

PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols $¢$ and $\$$. I think that this model is useless because *it is not able to recognize even finite languages*.
- We can generalize our model to a $k\text{-}RA_{\Delta nCL}$ which is the same model as the $k\text{-}RA_{\Delta CL}$ except that it uses n Δ symbols: Δ_1, Δ_2, ..., Δ_n. This model *is able to recognize each CFL*.
- We can study closure properties of these models.

PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols $¢$ and $\$$. I think that this model is useless because *it is not able to recognize even finite languages*.
- We can generalize our model to a $k\text{-}RA_{\Delta nCL}$ which is the same model as the $k\text{-}RA_{\Delta CL}$ except that it uses n Δ symbols: $\Delta_1, \Delta_2, ..., \Delta_n$. This model *is able to recognize each CFL*.
- We can study closure properties of these models.
- We can study decidability: $\mathrm{L(M)} = \emptyset$, $\mathrm{L(M)} = \Sigma^*$ etc.

PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols ¢ and $. I think that this model is useless because *it is not able to recognize even finite languages*.
- We can generalize our model to a $k\text{-}RA_{\Delta nCL}$ which is the same model as the $k\text{-}RA_{\Delta CL}$ except that it uses n Δ symbols: Δ_1, Δ_2, ..., Δ_n. This model *is able to recognize each CFL*.
- We can study closure properties of these models.
- We can study decidability: $L(M) = \emptyset$, $L(M) = \Sigma^*$ etc.
- We can study differences between language classes of RA_{CL} and $RA_{\Delta CL}$ (for different values of k) etc.

PART IV: Open Problems

- We can restrict our model to a $k\text{-}RA_{SIMPLE}$ which is the same model as the $k\text{-}RA_{CL}$ except that we do not use the symbols ¢ and $. I think that this model is useless because *it is not able to recognize even finite languages*.
- We can generalize our model to a $k\text{-}RA_{\Delta nCL}$ which is the same model as the $k\text{-}RA_{\Delta CL}$ except that it uses n Δ symbols: Δ_1, Δ_2, ..., Δ_n. This model *is able to recognize each CFL*.
- We can study closure properties of these models.
- We can study decidability: $L(M) = \emptyset$, $L(M) = \Sigma^*$ etc.
- We can study differences between language classes of RA_{CL} and $RA_{\Delta CL}$ (for different values of k) etc.
- We can study if these models are applicable in real problems: for example if we are able to recognize *Pascal* language etc.

References

- A. Cherubini, S. Crespi Reghizzi, and P.L. San Pietro: *Associative Language Descriptions*, Theoretical Computer Science, 270, 2002, 463-491.
- P. Jančar, F. Mráz, M. Plátek, J. Vogel: *On Monotonic Automata with a Restart Operation*. Journal of Automata, Languages and Combinatorics, 1999, 4(4):287–311.
- F. Mráz, F. Otto, M. Plátek: *Learning Analysis by Reduction from Positive Data. In: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E.* Tomita (Eds.): Proceedings ICGI 2006, LNCS 4201, Springer, Berlin, 2006, 125–136.
- **http://www.petercerno.wz.cz/ra.html**