Clearing Restarting Automata

PETER ČERNO
FRANTIŠEK MRÁZ

About

- We propose a new restricted version of restarting automata called **Clearing Restarting Automata**.

About

- We propose a new restricted version of restarting automata called **Clearing Restarting Automata**.
- The new model can be learned very efficiently from positive examples and its stronger version enables to learn effectively a large class of languages.

About

- We propose a new restricted version of restarting automata called **Clearing Restarting Automata**.
- The new model can be learned very efficiently from positive examples and its stronger version enables to learn effectively a large class of languages.
- We relate the class of languages recognized by clearing restarting automata to the Chomsky hierarchy.

Definition

- Let *k* be a *positive integer*.

Definition

- Let *k* be a *positive integer*.
- *k*-clearing restarting automaton (*k-cl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:

Definition

- Let k be a *positive integer*.
- k-clearing restarting automaton (k-cl-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - Σ is a finite nonempty *alphabet*, ¢, $\$ \notin \Sigma$.

Definition

- Let k be a *positive integer*.
- *k*-clearing restarting automaton (*k-cl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - 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 $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$

Definition

- Let k be a *positive integer*.
- k-clearing restarting automaton (k-cl-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - 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 $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$
  - The special symbols: $¢$ and $\$$ are called *sentinels*.

Definition

- Let k be a *positive integer*.
- *k*-clearing restarting automaton (*k-cl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - 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 $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$$
  - The special symbols: $¢$ and $\$$ are called *sentinels*.
  - The *width of the instruction* $i = (x, z, y)$ is $|i| = |xzy|$.

Definition

- A word $w = uzv$ can be *rewritten* to uv ($u\underline{z}v \vdash_M uv$) 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 $v.\$$)

Definition

- A word $w = uzv$ can be *rewritten* to uv ($u\underline{z}v \vdash_M uv$) 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 $w \vdash^*_M \lambda$ where $\vdash^*_M$ is reflexive and transitive closure of $\vdash_M$.

Definition

- A word $w = uzv$ can be *rewritten* to uv ($u\underline{z}v \vdash_M uv$) 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 $w \vdash^*_M \lambda$ where $\vdash^*_M$ is reflexive and transitive closure of $\vdash_M$.
- The *k-cl-RA*-automaton M *recognizes* the language $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Definition

- By *cl-RA* we will denote the class of all clearing restarting automata.

Definition

- By *cl-RA* we will denote the class of all clearing restarting automata.
- $\mathcal{L}$*(k-cl-RA)* denotes the class of all languages accepted by *k-cl-RA*-automata.

Definition

- By *cl-RA* we will denote the class of all clearing restarting automata.
- $\mathcal{L}(k\text{-}cl\text{-}RA)$ denotes the class of all languages accepted by *k-cl-RA*-automata.
- Similarly $\mathcal{L}(cl\text{-}RA)$ denotes the class of all languages accepted by *cl-RA*-automata.

Definition

- By *cl-RA* we will denote the class of all clearing restarting automata.
- $\mathcal{L}(k\text{-}cl\text{-}RA)$ denotes the class of all languages accepted by *k-cl-RA*-automata.
- Similarly $\mathcal{L}(cl\text{-}RA)$ denotes the class of all languages accepted by *cl-RA*-automata.
- $\mathcal{L}(cl\text{-}RA) = \bigcup_{k \geq 1} \mathcal{L}(k\text{-}cl\text{-}RA)$.

Definition

- By *cl-RA* we will denote the class of all clearing restarting automata.
- $\mathcal{L}(k\text{-}cl\text{-}RA)$ denotes the class of all languages accepted by *k-cl-RA*-automata.
- Similarly $\mathcal{L}(cl\text{-}RA)$ denotes the class of all languages accepted by *cl-RA*-automata.
- $\mathcal{L}(cl\text{-}RA) = \bigcup_{k \geq 1} \mathcal{L}(k\text{-}cl\text{-}RA)$.
- **Note**: For every *cl-RA* M: $\lambda \vdash^{*}_{M} \lambda$ hence $\lambda \in L(M)$. If we say that *cl-RA M* *recognizes a language L*, we mean that $L(M) = L \cup \{\lambda\}$.

Motivation

- 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

Motivation

- 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 simplicity of *cl-RA* model implies that the investigation of its properties is not so difficult and also the learning of languages is easy.

Motivation

- 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 simplicity of *cl-RA* model implies that the investigation of its properties is not so difficult and also the learning of languages is easy.
- Another important advantage of this model is that the instructions are human readable.

Example

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

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- Can be recognized by the *1-cl-RA* $M = (\{a, b\}, I)$, where the instructions I are:
  - $R1 = (a, \underline{ab}, b)$
  - $R2 = (¢, \underline{ab}, \$)$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- Can be recognized by the *1-cl-RA* $M = (\{a, b\}, I)$, where the instructions I are:
  - $R1 = (a, \underline{ab}, b)$
  - $R2 = (¢, \underline{ab}, \$)$
- For instance:
  - $aaa\underline{ab}bbb \vdash^{R1} aaabbb$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- Can be recognized by the *1-cl-RA* $M = (\{a, b\}, I)$, where the instructions I are:
  - $R1 = (a, \underline{ab}, b)$
  - $R2 = (¢, \underline{ab}, \$)$
- For instance:
  - $aaaabbbb \vdash^{R1} aa\underline{ab}bb \vdash^{R1} aabb$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- Can be recognized by the *1-cl-RA* $M = (\{a, b\}, I)$, where the instructions I are:
  - $R1 = (a, \underline{ab}, b)$
  - $R2 = (¢, \underline{ab}, \$)$
- For instance:
  - $aaaabbbb \vdash^{R1} aaabbb \vdash^{R1} a\underline{ab}b \vdash^{R1} ab$

Example

- Language $L = \{a^n b^n \mid n \geq 0\}$.
- Can be recognized by the *1-cl-RA* $M = (\{a, b\}, I)$, where the instructions I are:
  - $R1 = (a, \underline{ab}, b)$
  - $R2 = (¢, \underline{ab}, \$)$
- 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 $aaaabbbb \vdash^{*}_{M} \lambda$.

Some Theorems

- Error preserving property: Let $M = (\Sigma, I)$ be a *cl-RA-automaton* and u, v be two words from Σ^*. If $u \vdash^*_M v$ and $u \notin L(M)$, then $v \notin L(M)$.
  - **Proof.** $v \in L(M) \Rightarrow v \vdash^*_M \lambda \Rightarrow u \vdash^*_M v \vdash^*_M \lambda \Rightarrow u \in L(M)$. ■

Some Theorems

- Error preserving property: Let $M = (\Sigma, I)$ be a *cl-RA-*automaton and u, v be two words from Σ^*. If $u \vdash^*_M v$ and $u \notin L(M)$, then $v \notin L(M)$.
  - **Proof**. $v \in L(M) \Rightarrow v \vdash^*_M \lambda \Rightarrow u \vdash^*_M v \vdash^*_M \lambda \Rightarrow u \in L(M)$. ■
- Observation: For each finite $L \subseteq \Sigma^*$ there exist *1-cl-RA*-automaton M such that $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Suppose $L = \{w_1, \ldots, w_n\}$.
    Consider $I = \{(¢, w_1, \$), \ldots, (¢, w_n, \$)\}$. ■

Some Theorems

- Theorem: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.

Some Theorems

- Theorem: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.
- Theorem: For each regular language $L \subseteq \Sigma^*$ there exist a $k\text{-}cl\text{-}RA$-automaton M : $L(M) = L \cup \{\lambda\}$.

Some Theorems

- Theorem: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.
- Theorem: For each regular language $L \subseteq \Sigma^*$ there exist a $k\text{-}cl\text{-}RA$-automaton M : $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Based on *pumping lemma* for *regular languages*.

Some Theorems

- Theorem: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.
- Theorem: For each regular language $L \subseteq \Sigma^*$ there exist a $k\text{-}cl\text{-}RA$-automaton M: $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Based on *pumping lemma* for *regular languages*.
  - For each $z \in \Sigma^*$, $|z|=n$ there exist u, v, w such that $|v| \geq 1$ and $\delta(q_0, uv) = \delta(q_0, u)$; the word v can be crossed out.

Some Theorems

- Theorem: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.
- Theorem: For each regular language $L \subseteq \Sigma^*$ there exist a k-cl-RA-automaton M : $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Based on *pumping lemma* for *regular languages*.
  - For each $z \in \Sigma^*$, $|z|=n$ there exist u, v, w such that $|v| \geq 1$ and $\delta(q_0, uv) = \delta(q_0, u)$; the word v can be crossed out.
  - We add corresponding instruction $i_z = (¢.u, v, w)$.

Some Theorems

- <u>Theorem</u>: $\mathcal{L}(k\text{-}cl\text{-}RA) \subset \mathcal{L}((k+1)\text{-}cl\text{-}RA)$, for all $k \geq 1$.
  - **Note**: The following language: $\{ (c^k a c^k)^n (c^k b c^k)^n \mid n \geq 0 \}$ belongs to $\mathcal{L}((k+1)\text{-}cl\text{-}RA) - \mathcal{L}(k\text{-}cl\text{-}RA)$.
- <u>Theorem</u>: For each regular language $L \subseteq \Sigma^*$ there exist a k-cl-RA-automaton M : $L(M) = L \cup \{\lambda\}$.
  - **Proof**. Based on *pumping lemma* for *regular languages*.
  - For each $z \in \Sigma^*$, $|z|=n$ there exist u, v, w such that $|v| \geq 1$ and $\delta(q_0, uv) = \delta(q_0, u)$; the word v can be crossed out.
  - We add corresponding instruction $i_z = (¢.u, v, w)$.
  - For each accepted $z \in \Sigma^{<n} - \{\lambda\}$ we add instruction $i_z = (¢, z, \$)$.

Some Theorems

- Theorem: The language $L_1 = \{a^n cb^n \mid n \geq 0\} \cup \{\lambda\}$ is not recognized by any *cl-RA*-automaton.

Some Theorems

- Theorem: The language $L_1 = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$ is not recognized by any *cl-RA*-automaton.
  - **Note**: L_1 can be recognized by a simple *RRWW*-automaton. Moreover L_1 is a *context-free language*, thus we get the following corollary:
- Corollary:
  - $\mathcal{L}(cl\text{-}RA) \subset \mathcal{L}(RRWW)$.
  - CFL - $\mathcal{L}(cl\text{-}RA) \neq \emptyset$.

Some Theorems

- Let $L_2 = \{a^n b^n \mid n \geq 0\}$ and $L_3 = \{a^n b^{2n} \mid n \geq 0\}$ be two sample languages. Apparently both L_2 and L_3 are recognized by *1-cl-RA*-automata.

Some Theorems

- Let $L_2 = \{a^n b^n \mid n \geq 0\}$ and $L_3 = \{a^n b^{2n} \mid n \geq 0\}$ be two sample languages. Apparently both L_2 and L_3 are recognized by *1-cl-RA*-automata.
- Theorem: Languages $L_2 \cup L_3$ and $L_2 . L_3$ are not recognized by any *cl-RA*-automaton.

Some Theorems

- Let $L_2 = \{a^n b^n \mid n \geq 0\}$ and $L_3 = \{a^n b^{2n} \mid n \geq 0\}$ be two sample languages. Apparently both L_2 and L_3 are recognized by *1-cl-RA*-automata.
- Theorem: Languages $L_2 \cup L_3$ and $L_2 . L_3$ are not recognized by any *cl-RA*-automaton.
- Corollary: $\mathcal{L}(cl\text{-}RA)$ is not closed under union, concatenation, and homomorphism.
  - For homomorphism use $\{a^n b^n \mid n \geq 0\} \cup \{c^n d^{2n} \mid n \geq 0\}$ and homomorphism defined as: $a \mapsto a$, $b \mapsto b$, $c \mapsto a$, $d \mapsto b$. ■

Some Theorems

- It is easy to see that each of the following languages:
  - $L_4 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_5 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_6 = \{a^m b^m \mid m \geq 0\}$

  can be recognized by a *1-cl-RA*-automaton.

Some Theorems

- It is easy to see that each of the following languages:
  - $L_4 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_5 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_6 = \{a^m b^m \mid m \geq 0\}$

  can be recognized by a *1-cl-RA*-automaton.
- Corollary: $\mathcal{L}(cl\text{-}RA)$ is not closed under:
  - intersection: $L_1 = L_4 \cap L_5$.

Some Theorems

- It is easy to see that each of the following languages:
  - $L_4 = \{a^n cb^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_5 = \{a^n cb^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_6 = \{a^m b^m \mid m \geq 0\}$

  can be recognized by a *1-cl-RA*-automaton.
- Corollary: $\mathcal{L}(cl\text{-}RA)$ is not closed under:
  - intersection: $L_1 = L_4 \cap L_5$.
  - intersection with regular language: L_5 is regular.

Some Theorems

- It is easy to see that each of the following languages:
  - $L_4 = \{a^n c b^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_5 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_6 = \{a^m b^m \mid m \geq 0\}$

  can be recognized by a *1-cl-RA*-automaton.
- <u>Corollary</u>: $\mathcal{L}(cl\text{-}RA)$ is not closed under:
  - intersection: $L_1 = L_4 \cap L_5$.
  - intersection with regular language: L_5 is regular.
  - set difference: $L_1 = (L_4 - L_6) \cup \{\lambda\}$.

Parentheses

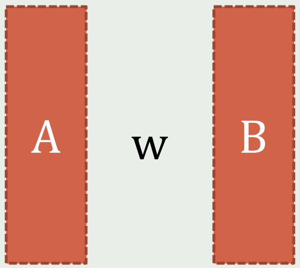
- The following instruction of *1-cl-RA M* is enough for recognizing the language of correct parentheses:
- $(\lambda, (\,), \lambda)$

Parentheses

- The following instruction of *1-cl-RA 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-cl-RA 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 $\Sigma = \{a, +, *, (,)\}$.

Arithmetic Expressions

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

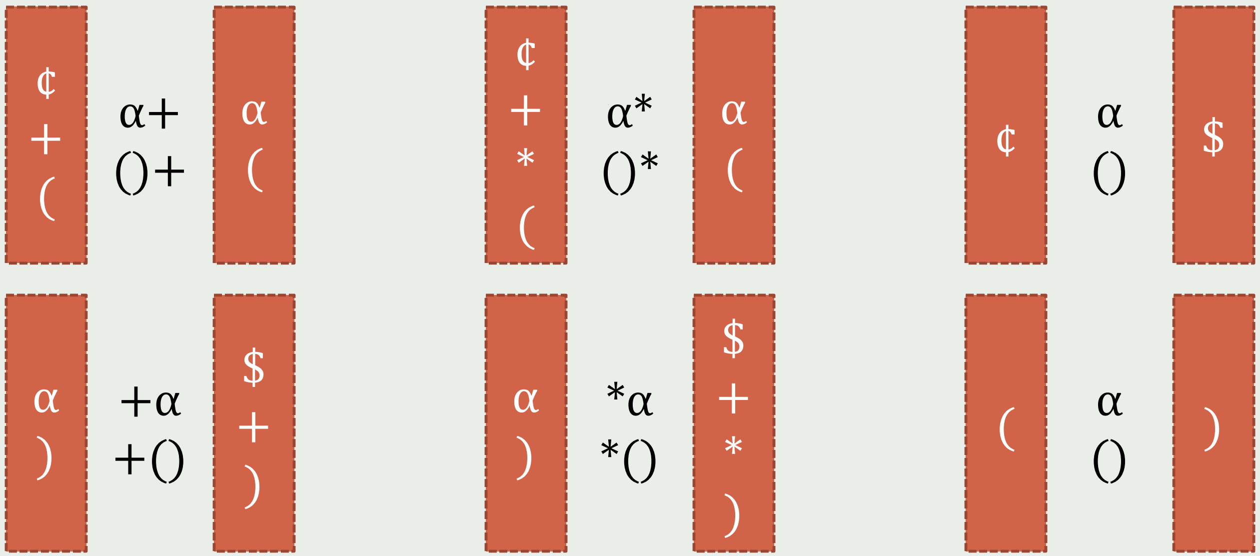
Arithmetic Expressions

- 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-cl-RA*-automaton is sufficient:



Arithmetic Expressions - Example

Expression	Instruction
α*α + ((α + α) + (α + α*α))*α	(¢, α*, α)
α + ((α + α) + (α + α*α))*α	(α, +α,))
α + ((α) + (α + α*α))*α	( ), *α, $)
α + ((α) + (α + α*α))	(+, α*, α)
α + ((α) + (α + α))	((, α+, α)
α + ((α) + (α))	((, α,))
α + (( ) + (α))	((, ( )+, ( )
α + ((α))	((, α,))
α + (( ))	((, ( ),))
α + ( )	(¢, α+, ( )
( )	(¢, ( ), $)
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:
  - $R1 = (bb, \underline{a}, bbbb)$
  - $R2 = (bb, \underline{bb}, \$)$
  - $R3 = (¢, \underline{cbb}, \$)$

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

Nondeterminism

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

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

  then it would not be possible to continue.

Nondeterminism

- Assume the following instructions:
  - $R1 = (bb, \underline{a}, bbbb)$
  - $R2 = (bb, \underline{bb}, \$)$
  - $R3 = (¢, \underline{cbb}, \$)$

  and the word: *cbbabbbb*. Then:
  - $cbb\underline{a}bbbb \vdash^{R1} cbbbb\underline{bb} \vdash^{R2} cbb\underline{bb} \vdash^{R2} \underline{cbb} \vdash^{R3} \lambda$.
- **But** if we have started with *R2*:
  - $cbbabb\underline{bb} \vdash^{R2} cbbabb$

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

Greibach's Hardest CFL

- As we have seen not all context-free languages are recognized by a *cl-RA*-automaton.

Greibach's Hardest CFL

- As we have seen not all context-free languages are recognized by a *cl-RA*-automaton.
- We still can characterize CFL using clearing restarting automata, inverse homomorphism and Greibach's hardest context-free language.

Greibach's Hardest CFL

- Greibach constructed a context-free language *H*, such that:
  - Any context-free language can be parsed in whatever time or space it takes to recognize *H*.

Greibach's Hardest CFL

- Greibach constructed a context-free language *H*, such that:
  - Any context-free language can be parsed in whatever time or space it takes to recognize *H*.
  - Any context-free language L can be obtained from *H* by an inverse homomorphism. That is, for each context-free language *L*, there exists a homomorphism φ: $L = \varphi^{-1}(H)$.

Greibach's Hardest CFL

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}, d \notin \Sigma$.

Greibach's Hardest CFL

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}, d \notin \Sigma$.
- Let D_2 be *Semi-Dyck language* on $\{a_1, a_2, \underline{a}_1, \underline{a}_2\}$ generated by the grammar: $S \rightarrow \lambda \mid SS \mid a_1 S \underline{a}_1 \mid a_2 S \underline{a}_2$.

Greibach's Hardest CFL

- By *S. A. Greibach*, definition from *Section 10.5* of *M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.*
- Let $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$, $d \notin \Sigma$.
- Let D_2 be *Semi-Dyck language* on $\{a_1, a_2, \underline{a}_1, \underline{a}_2\}$ generated by the grammar: $S \rightarrow \lambda \mid SS \mid a_1 S \underline{a}_1 \mid a_2 S \underline{a}_2$.
- Then $H = \{\lambda\} \cup \{\prod_{i=1..n} x_i c y_i c z_i d \mid n \geq 1, y_1 y_2 \ldots y_n \in \#D_2, x_i, z_i \in \Sigma^*\}$,
  - $y_1 \in \# . \{a_1, a_2, \underline{a}_1, \underline{a}_2\}^*$,
  - $y_i \in \{a_1, a_2, \underline{a}_1, \underline{a}_2\}^*$ for all $i > 1$.

Greibach's Hardest CFL

- Theorem: H is not accepted by any *cl-RA*-automaton.

Greibach's Hardest CFL

- Theorem: *H* is not accepted by any *cl-RA*-automaton.
- Cherubini et. al defined *H* using *associative language description (ALD)* which uses one auxiliary symbol.

  (*in Associative language descriptions, Theoretical Computer Science, 270 (2002), 463-491*)

Greibach's Hardest CFL

- Theorem: *H* is not accepted by any *cl-RA*-automaton.
- Cherubini et. al defined *H* using *associative language description (ALD)* which uses one auxiliary symbol.

  (*in Associative language descriptions, Theoretical Computer Science, 270 (2002), 463-491*)
- So we will slightly extend the definition of *cl-RA*-automata in order to be able to recognize more languages including *H*.

Definition

- Let k be a *positive integer*.

Definition

- Let *k* be a *positive integer*.
- *k-Δ*-clearing restarting automaton (*k-Δcl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:

Definition

- Let *k* be a *positive integer*.
- *k-Δ*-clearing restarting automaton (*k-Δcl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - Σ is a finite nonempty alphabet, $¢, \$, \Delta \notin \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$.

Definition

- Let k be a *positive integer*.
- *k-Δ*-clearing restarting automaton (*k-Δcl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - Σ is a finite nonempty alphabet, $¢, \$, \Delta \notin \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$.
  - I is a finite set of instructions of the following forms:
    - (1) $(x, z \to \lambda, y)$
    - (2) $(x, z \to \Delta, y)$

Definition

- Let k be a *positive integer*.
- *k-Δ*-clearing restarting automaton (*k-Δcl-RA*-automaton for short) is a couple $M = (\Sigma, I)$, where:
  - Σ is a finite nonempty alphabet, $¢, \$, \Delta \notin \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$.
  - I is a finite set of instructions of the following forms:
    - (1) $(x, z \to \lambda, y)$
    - (2) $(x, z \to \Delta, y)$
  - 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 $RC_k = \Gamma^k \cup \Gamma^{\leq k-1}.\$$

Definition

- A word $w = uzv$ can be *rewritten* to usv ($u\underline{z}v \vdash_M usv$) if and only if there exist an instruction $i = (x, z \rightarrow s, y) \in I$ such that:
  - $x \sqsupseteq ¢.u$ (x is a suffix of $¢.u$)
  - $y \sqsubseteq v.\$$ (y is a prefix of $v.\$$)

Definition

- A word $w = uzv$ can be *rewritten* to usv ($u\underline{z}v \vdash_M usv$) if and only if there exist an instruction $i = (x, z \rightarrow s, y) \in I$ 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 $w \vdash^{*}_{M} \lambda$ where $\vdash^{*}_{M}$ is reflexive and transitive closure of $\vdash_M$.

Definition

- A word $w = uzv$ can be *rewritten* to usv ($uzv \vdash_M usv$) if and only if there exist an instruction $i = (x, z \rightarrow s, y) \in I$ 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 $w \vdash^*_M \lambda$ where $\vdash^*_M$ is reflexive and transitive closure of $\vdash_M$.
- The *k-Δcl-RA*-automaton M *recognizes* the language $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Definition

- By *Δcl-RA* we will denote the class of all *Δ*- clearing restarting automata.

Definition

- By *Δcl-RA* we will denote the class of all *Δ-* clearing restarting automata.
- *$\mathcal{L}$(k-Δcl-RA)* denotes the class of all languages accepted by *k-Δcl-RA*-automata.

Definition

- By *Δcl-RA* we will denote the class of all *Δ-* clearing restarting automata.
- *ℒ(k-Δcl-RA)* denotes the class of all languages accepted by *k-Δcl-RA*-automata.
- Similarly *ℒ(Δcl-RA)* denotes the class of all languages accepted by *Δcl-RA*-automata.

Definition

- By *Δcl-RA* we will denote the class of all *Δ*- clearing restarting automata.
- *ℒ(k-Δcl-RA)* denotes the class of all languages accepted by *k-Δcl-RA*-automata.
- Similarly *ℒ(Δcl-RA)* denotes the class of all languages accepted by *Δcl-RA*-automata.
- $\mathcal{L}(\Delta cl\text{-}RA) = \bigcup_{k\geq 1}\mathcal{L}(k\text{-}\Delta cl\text{-}RA)$.

Definition

- By $\Delta cl\text{-}RA$ we will denote the class of all Δ- clearing restarting automata.
- $\mathcal{L}(k\text{-}\Delta cl\text{-}RA)$ denotes the class of all languages accepted by $k\text{-}\Delta cl\text{-}RA$-automata.
- Similarly $\mathcal{L}(\Delta cl\text{-}RA)$ denotes the class of all languages accepted by $\Delta cl\text{-}RA$-automata.
- $\mathcal{L}(\Delta cl\text{-}RA) = \bigcup_{k \geq 1} \mathcal{L}(k\text{-}\Delta cl\text{-}RA)$.
- **Note**: For every $\Delta cl\text{-}RA$ M: $\lambda \vdash^{*}_{M} \lambda$ hence $\lambda \in L(M)$. If we say that $\Delta cl\text{-}RA$ M recognizes a language L, we mean that $L(M) = L \cup \{\lambda\}$.

Example

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

Example

- Language $L = \{a^n c b^n \mid n \geq 0\}$.
- Can be recognized by the *1-Δcl-RA* $M = (\{a, b, c\}, I)$, where the instructions I are:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (¢, c \rightarrow \lambda, \$)$
  - $R\Delta1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta2 = (¢, a\Delta b \rightarrow \lambda, \$)$

Example

- Language $L = \{a^n c b^n \mid n \geq 0\}$.
- Can be recognized by the $1\text{-}\Delta cl\text{-}RA$ $M = (\{a, b, c\}, I)$, where the instructions I are:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (¢, c \rightarrow \lambda, \$)$
  - $R\Delta 1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta 2 = (¢, a\Delta b \rightarrow \lambda, \$)$
- For instance:
  - $aaa\underline{c}bbb \vdash^{Rc1} aa\Delta bb$

Example

- Language $L = \{a^n c b^n \mid n \geq 0\}$.
- Can be recognized by the *1-Δcl-RA* $M = (\{a, b, c\}, I)$, where the instructions I are:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (¢, c \rightarrow \lambda, \$)$
  - $R\Delta 1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta 2 = (¢, a\Delta b \rightarrow \lambda, \$)$
- For instance:
  - $aaacbbb \vdash^{Rc1} a\underline{a\Delta b}b \vdash^{R\Delta 1} a\Delta b$

Example

- Language $L = \{a^n c b^n \mid n \geq 0\}$.
- Can be recognized by the *1-Δcl-RA* $M = (\{a, b, c\}, I)$, where the instructions I are:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (¢, c \rightarrow \lambda, \$)$
  - $R\Delta 1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta 2 = (¢, a\Delta b \rightarrow \lambda, \$)$
- For instance:
  - $aaacbbb \vdash^{Rc1} aa\Delta bb \vdash^{R\Delta 1} \underline{a\Delta b} \vdash^{R\Delta 2} \lambda$.
- Now we see that the word *aaacbbb* is accepted because $aaacbbb \vdash^{*}_{M} \lambda$.

Back to Greibach's Hardest CFL

- Theorem: Greibach's Hardest CFL *H* is recognized by a *1-Δcl-RA*-automaton.

Back to Greibach's Hardest CFL

- <u>Theorem</u>: Greibach's Hardest CFL H is recognized by a *1-Δcl-RA*-automaton.
  - **Idea**. Suppose that we have $w \in H$:

    $w = ¢\, x_1cy_1cz_1d\ \ x_2cy_2cz_2d \ldots x_ncy_ncz_nd\ \$$

Back to Greibach's Hardest CFL

- Theorem: Greibach's Hardest CFL *H* is recognized by a *1-Δcl-RA*-automaton.
  - **Idea**. Suppose that we have $w \in H$:

    $w = ¢\, x_1cy_1cz_1d\ \ x_2cy_2cz_2d \ldots x_ncy_ncz_nd\ \$$
  - In the *first phase* we start with deleting letters (from the alphabet $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$) from the right side of ¢ and from the left and right sides of the letters *d*.

Back to Greibach's Hardest CFL

- <u>Theorem</u>: Greibach's Hardest CFL H is recognized by a *1-Δcl-RA*-automaton.
  - **Idea**. Suppose that we have $w \in H$:

    $w = ¢\, x_1cy_1cz_1d\ \ x_2cy_2cz_2d \ldots x_ncy_ncz_nd\ \$$
  - In the *first phase* we start with deleting letters (from the alphabet $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$) 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:

    $¢\, cy_1cd\ \ cy_2cd \ldots cy_ncd\ \$$, we introduce the Δ symbols:

    $¢\, \Delta y_1 \Delta y_2 \Delta \ldots \Delta y_n \Delta\ \$$

Back to Greibach's Hardest CFL

- <u>Theorem</u>: Greibach's Hardest CFL H is recognized by a *1-Δcl-RA*-automaton.
  - **Idea**. Suppose that we have $w \in H$:

    $w = ¢\, x_1cy_1cz_1d\ \ x_2cy_2cz_2d \ldots x_ncy_ncz_nd\ \$$
  - In the *first phase* we start with deleting letters (from the alphabet $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$) 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:

    $¢\, cy_1cd\ \ cy_2cd \ldots cy_ncd\ \$$, we introduce the Δ symbols:

    $¢\, \Delta y_1 \Delta y_2 \Delta \ldots \Delta y_n \Delta\ \$$
  - In the *second phase* we check if $y_1y_2 \ldots y_n \in \#D_2$.

Instructions recognizing Hardest CFL H

- Suppose $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$, $d \notin \Sigma$, $\Gamma = \Sigma \cup \{d, \Delta\}$.

Instructions for the first phase:	Instructions for the second phase:
(1) $(¢, \Sigma \to \lambda, \Sigma)$	(7) $(\Gamma, a_1\underline{a}_1 \to \lambda, \Gamma - \{\#\})$
(2) $(\Sigma, \Sigma \to \lambda, d)$	(8) $(\Gamma, a_2\underline{a}_2 \to \lambda, \Gamma - \{\#\})$
(3) $(d, \Sigma \to \lambda, \Sigma)$	(9) $(\Gamma, a_1\Delta\underline{a}_1 \to \Delta, \Gamma - \{\#\})$
(4) $(¢, c \to \Delta, \Sigma \cup \{\Delta\})$	(10) $(\Gamma, a_2\Delta\underline{a}_2 \to \Delta, \Gamma - \{\#\})$
(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) $(¢, \Delta\#\Delta \to \lambda, \$)$

Instructions recognizing Hardest CFL H

- Suppose $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$, $d \notin \Sigma$, $\Gamma = \Sigma \cup \{d, \Delta\}$.

Instructions for the first phase:	Instructions for the second phase:
(1) $(¢, \Sigma \rightarrow \lambda, \Sigma)$	(7) $(\Gamma, a_1\underline{a}_1 \rightarrow \lambda, \Gamma - \{\#\})$
(2) $(\Sigma, \Sigma \rightarrow \lambda, d)$	(8) $(\Gamma, a_2\underline{a}_2 \rightarrow \lambda, \Gamma - \{\#\})$
(3) $(d, \Sigma \rightarrow \lambda, \Sigma)$	(9) $(\Gamma, a_1\Delta\underline{a}_1 \rightarrow \Delta, \Gamma - \{\#\})$
(4) $(¢, c \rightarrow \Delta, \Sigma \cup \{\Delta\})$	(10) $(\Gamma, a_2\Delta\underline{a}_2 \rightarrow \Delta, \Gamma - \{\#\})$
(5) $(\Sigma \cup \{\Delta\}, cdc \rightarrow \Delta, \Sigma \cup \{\Delta\})$	(11) $(\Sigma - \{c\}, \Delta \rightarrow \lambda, \Delta)$
(6) $(\Sigma \cup \{\Delta\}, cd \rightarrow \Delta, \$)$	(12) $(¢, \Delta\#\Delta \rightarrow \lambda, \$)$

- In fact, there is no such thing as a *first phase* or a *second phase*. We have only instructions.

Instructions recognizing Hardest CFL H

- Suppose $\Sigma = \{a_1, a_2, \underline{a}_1, \underline{a}_2, \#, c\}$, $d \notin \Sigma$, $\Gamma = \Sigma \cup \{d, \Delta\}$.

Instructions for the first phase:	**Instructions for the second phase:**
(1) $(¢, \Sigma \to \lambda, \Sigma)$	(7) $(\Gamma, a_1\underline{a}_1 \to \lambda, \Gamma - \{\#\})$
(2) $(\Sigma, \Sigma \to \lambda, d)$	(8) $(\Gamma, a_2\underline{a}_2 \to \lambda, \Gamma - \{\#\})$
(3) $(d, \Sigma \to \lambda, \Sigma)$	(9) $(\Gamma, a_1\Delta\underline{a}_1 \to \Delta, \Gamma - \{\#\})$
(4) $(¢, c \to \Delta, \Sigma \cup \{\Delta\})$	(10) $(\Gamma, a_2\Delta\underline{a}_2 \to \Delta, \Gamma - \{\#\})$
(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) $(¢, \Delta\#\Delta \to \lambda, \$)$

- In fact, there is no such thing as a *first phase* or a *second phase*. We have only instructions.
- Theorem: $H \subseteq L(M)$, $H \supseteq L(M)$.

Learning Clearing Restarting Automata

- Let $u_i \vdash_M v_i$, $i = 1,2 \ldots, n$ be a list of known reductions.

Learning Clearing Restarting Automata

- Let $u_i \vdash_M v_i$, $i = 1,2 \ldots, n$ be a list of known reductions.
- An algorithm for machine learning the unknown clearing restarting automaton can be outlined as follows:

Learning Clearing Restarting Automata

- Let $u_i \vdash_M v_i$, $i = 1,2 \ldots, n$ be a list of known reductions.
- An algorithm for machine learning the unknown clearing restarting automaton can be outlined as follows:

  Step 1: $k := 1$.

Learning Clearing Restarting Automata

- Let $u_i \vdash_M v_i$, $i = 1,2 \ldots, n$ be a list of known reductions.
- An algorithm for machine learning the unknown clearing restarting automaton can be outlined as follows:

Step 1: $k := 1$.

Step 2: For each reduction $u_i \vdash_M v_i$ choose (nondeterministically) a factorization of u_i, such that $u_i = x_i z_i y_i$ and $v_i = x_i y_i$.

Learning Clearing Restarting Automata

Step 3: Construct a k-cl-RA-automaton $M = (\Sigma, I)$, where $I = \{ (Suff_k(¢.x_i), z_i, Pref_k(y_i.\$)) \mid i = 1, \ldots, n \}$.

Learning Clearing Restarting Automata

<u>Step 3</u>: Construct a k-cl-RA-automaton $M = (\Sigma, I)$, where $I = \{ (Suff_k(¢.x_i), z_i, Pref_k(y_i.\$)) \mid i = 1, \ldots, n \}$.

- $Pref_k(u)$ ($Suff_k(u)$, resp.) denotes the *prefix* (*suffix* , resp.) of length k of the string u in case $|u| > k$, or the whole u in case $|u| \leq k$.

Learning Clearing Restarting Automata

Step 3: Construct a *k-cl-RA*-automaton $M = (\Sigma, I)$, where $I = \{ (Suff_k(¢.x_i), z_i, Pref_k(y_i.\$)) \mid i = 1, \ldots, n \}$.

- $Pref_k(u)$ ($Suff_k(u)$, resp.) denotes the *prefix* (*suffix* , resp.) of length k of the string u in case $|u| > k$, or the whole u in case $|u| \leq k$.

Step 4: Test the automaton *M* using any available information e.g. some negative samples of words.

Learning Clearing Restarting Automata

<u>Step 3</u>: Construct a *k-cl-RA*-automaton $M = (\Sigma, I)$, where $I = \{ (Suff_k(¢.x_i), z_i, Pref_k(y_i.\$)) \mid i = 1, \ldots, n \}$.

- $Pref_k(u)$ ($Suff_k(u)$, resp.) denotes the *prefix* (*suffix* , resp.) of length k of the string u in case $|u| > k$, or the whole u in case $|u| \leq k$.

<u>Step 4</u>: Test the automaton M using any available information e.g. some negative samples of words.

<u>Step 5</u>: If the automaton passed all the tests, return M. Otherwise try another factorization of the known reductions and continue by Step 3 or increase k and continue by Step 2.

Learning Clearing Restarting Automata

- Even if the algorithm is very simple, it can be used to infer some non-trivial clearing (and after some generalization also Δ-clearing) restarting automata.

Learning Clearing Restarting Automata

- Even if the algorithm is very simple, it can be used to infer some non-trivial clearing (and after some generalization also Δ-clearing) restarting automata.
- Although Δ-clearing restarting automata are stronger than clearing restarting automata, we will see that even clearing restarting automata can recognize some *non-context-free languages*.

Learning Clearing Restarting Automata

- Even if the algorithm is very simple, it can be used to infer some non-trivial clearing (and after some generalization also Δ-clearing) restarting automata.
- Although Δ-clearing restarting automata are stronger than clearing restarting automata, we will see that even clearing restarting automata can recognize some *non-context-free languages*.
- However, it can be shown, that:
- Theorem: $\mathcal{L}(\Delta cl\text{-}RA) \subseteq$ CSL, where CSL denotes the class of *context-sensitive languages*.

Learning Non-Context-Free Language

- Theorem: There exists a *k-cl-RA*-automaton *M* recognizing a language that is not context-free.

Learning Non-Context-Free Language

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

Learning Non-Context-Free Language

- <u>Theorem</u>: There exists a *k-cl-RA*-automaton *M* recognizing a language that is not context-free.
  - **Idea**. We try to create a *k-cl-RA*-automaton *M* such that $L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2^m} \mid m\geq 0\}$.
  - 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.

Learning Non-Context-Free Language

- Example:

  ¢ abababababababab $

Learning Non-Context-Free Language

- <u>Example</u>:

  $¢\ ababababababab\underline{a}b\ \$ \vdash_M ¢\ ababababab\underline{a}babb\ \$ \vdash_M$

  $¢\ ababab\underline{a}babbabb\ \$ \vdash_M ¢\ ab\underline{a}babbabbabb\ \$ \vdash_M$

  $¢\ abbabbabbabb\ \$$

Learning Non-Context-Free Language

- <u>Example</u>:

  $¢\ ababababababab\underline{a}b\ \$ \vdash_M ¢\ ababababab\underline{a}babb\ \$ \vdash_M$

  $¢\ ababab\underline{a}babbabb\ \$ \vdash_M ¢\ ab\underline{a}babbabbabb\ \$ \vdash_M$

  $¢\ abbabbabba\underline{b}b\ \$ \vdash_M ¢\ abbabba\underline{b}bab\ \$ \vdash_M$

  $¢\ abba\underline{b}babab\ \$ \vdash_M ¢\ a\underline{b}bababab\ \$ \vdash_M$

  $¢\ abababab\ \$$

Learning Non-Context-Free Language

- Example:

  $¢\ ababababababab\underline{a}b\ \$ \vdash_M ¢\ ababababab\underline{a}babb\ \$ \vdash_M$

  $¢\ ababab\underline{a}babbabb\ \$ \vdash_M ¢\ ab\underline{a}babbabbabb\ \$ \vdash_M$

  $¢\ abbabbabba\underline{b}b\ \$ \vdash_M ¢\ abbabba\underline{b}bab\ \$ \vdash_M$

  $¢\ abba\underline{b}babab\ \$ \vdash_M ¢\ a\underline{b}bababab\ \$ \vdash_M$

  $¢\ ababab\underline{a}b\ \$ \vdash_M ¢\ ab\underline{a}babb\ \$ \vdash_M$

  $¢\ abba\underline{b}b\ \$$

Learning Non-Context-Free Language

- Example:

¢ $ababababababab\underline{a}b$ \$ $\vdash_M$ ¢ $ababababab\underline{a}babb$ \$ $\vdash_M$

¢ $ababab\underline{a}babbabb$ \$ $\vdash_M$ ¢ $ab\underline{a}babbabbabb$ \$ $\vdash_M$

¢ $abbabbabba\underline{b}b$ \$ $\vdash_M$ ¢ $abbabba\underline{b}bab$ \$ $\vdash_M$

¢ $abba\underline{b}babab$ \$ $\vdash_M$ ¢ $a\underline{b}bababab$ \$ $\vdash_M$

¢ $ababab\underline{a}b$ \$ $\vdash_M$ ¢ $ab\underline{a}babb$ \$ $\vdash_M$

¢ $abba\underline{b}b$ \$ $\vdash_M$ ¢ $a\underline{b}bab$ \$ $\vdash_M$

¢ $abab$ \$

Learning Non-Context-Free Language

- <u>Example</u>:

$¢\,ababababababab\underline{a}b\,\$ \vdash_M ¢\,abababababa\underline{a}babb\,\$ \vdash_M$

$¢\,ababab\underline{a}babbabb\,\$ \vdash_M ¢\,ab\underline{a}babbabbabb\,\$ \vdash_M$

$¢\,abbabbabba\underline{b}b\,\$ \vdash_M ¢\,abbabba\underline{b}bab\,\$ \vdash_M$

$¢\,abba\underline{b}babab\,\$ \vdash_M ¢\,a\underline{b}bababab\,\$ \vdash_M$

$¢\,ababab\underline{a}b\,\$ \vdash_M ¢\,ab\underline{a}babb\,\$ \vdash_M$

$¢\,abba\underline{b}b\,\$ \vdash_M ¢\,a\underline{b}bab\,\$ \vdash_M$

$¢\,ab\underline{a}b\,\$ \vdash_M ¢\,a\underline{b}b\,\$ \vdash_M ¢\,\underline{ab}\,\$ \vdash_M ¢\,\lambda\,\$$ *accept*.

Learning Non-Context-Free Language

- <u>Example</u>:

  $¢\ ababababababab\underline{a}b\ \$ \vdash_M ¢\ ababababab\underline{a}babb\ \$ \vdash_M$

  $¢\ ababab\underline{a}babbabb\ \$ \vdash_M ¢\ ab\underline{a}babbabbabb\ \$ \vdash_M$

  $¢\ abbabbabba\underline{b}b\ \$ \vdash_M ¢\ abbabba\underline{b}bab\ \$ \vdash_M$

  $¢\ abba\underline{b}babab\ \$ \vdash_M ¢\ a\underline{b}bababab\ \$ \vdash_M$

  $¢\ ababab\underline{a}b\ \$ \vdash_M ¢\ ab\underline{a}babb\ \$ \vdash_M$

  $¢\ abba\underline{b}b\ \$ \vdash_M ¢\ a\underline{b}bab\ \$ \vdash_M$

  $¢\ ab\underline{a}b\ \$ \vdash_M ¢\ a\underline{b}b\ \$ \vdash_M ¢\ \underline{ab}\ \$ \vdash_M ¢\ \lambda\ \$$ *accept*.

- From this sample computation we can collect 15 reductions with unambiguous factorizations and use them as an input to our algorithm.

Learning Non-Context-Free Language

- The only variable we have to choose is k - the length of the context of the instructions.

Learning Non-Context-Free Language

- The only variable we have to choose is *k* - the length of the context of the instructions.
- For $k = 1$ we get the following set of instructions:

  (b, a, b), (a, b, b), (¢, ab, $)

Learning Non-Context-Free Language

- The only variable we have to choose is k - the length of the context of the instructions.
- For $k = 1$ we get the following set of instructions:

  $(b, \underline{a}, b), (a, \underline{b}, b), (¢, \underline{ab}, \$)$

  But then the automaton would accept the word *ababab* which does not belong to L:

  $abab\underline{a}b \vdash_M ab\underline{a}bb \vdash_M a\underline{b}bb \vdash_M a\underline{b}b \vdash_M \underline{ab} \vdash_M \lambda.$

Learning Non-Context-Free Language

- For $k = 2$ we get the following set of instructions:

  $(ab, \underline{a}, \{b\$, ba\}), (\{¢a, ba\}, \underline{b}, \{b\$, ba\}), (¢, \underline{ab}, \$)$

Learning Non-Context-Free Language

- For $k = 2$ we get the following set of instructions:

  $(ab, \underline{a}, \{b\$, ba\}), (\{¢a, ba\}, \underline{b}, \{b\$, ba\}), (¢, \underline{ab}, \$)$

  But then the automaton would accept the word *ababab* which does not belong to L:

  $abab\underline{a}b \vdash_M aba\underline{b}b \vdash_M ab\underline{a}b \vdash_M a\underline{b}b \vdash_M \underline{ab} \vdash_M \lambda.$

Learning Non-Context-Free Language

- For *k = 2* we get the following set of instructions:

  (ab, <u>a</u>, {b\$, ba}), ({¢a, ba}, <u>b</u>, {b\$, ba}), (¢, <u>ab</u>, \$)

  But then the automaton would accept the word *ababab* which does not belong to *L*:

  $aba\underline{b}ab \vdash_M aba\underline{b}b \vdash_M ab\underline{a}b \vdash_M a\underline{b}b \vdash_M \underline{ab} \vdash_M \lambda.$

- For *k = 3* we get the following set of instructions:

  ({¢ab, bab}, <u>a</u>, {b\$, bab}), ({¢a, bba}, <u>b</u>, {b\$, bab}), (¢, <u>ab</u>, \$)

Learning Non-Context-Free Language

- For $k = 2$ we get the following set of instructions:

  $(ab, \underline{a}, \{b\$, ba\}), (\{\text{¢}a, ba\}, \underline{b}, \{b\$, ba\}), (\text{¢}, \underline{ab}, \$)$

  But then the automaton would accept the word *ababab* which does not belong to L:

  $abab\underline{a}b \vdash_M aba\underline{b}b \vdash_M ab\underline{a}b \vdash_M a\underline{b}b \vdash_M \underline{ab} \vdash_M \lambda.$

- For $k = 3$ we get the following set of instructions:

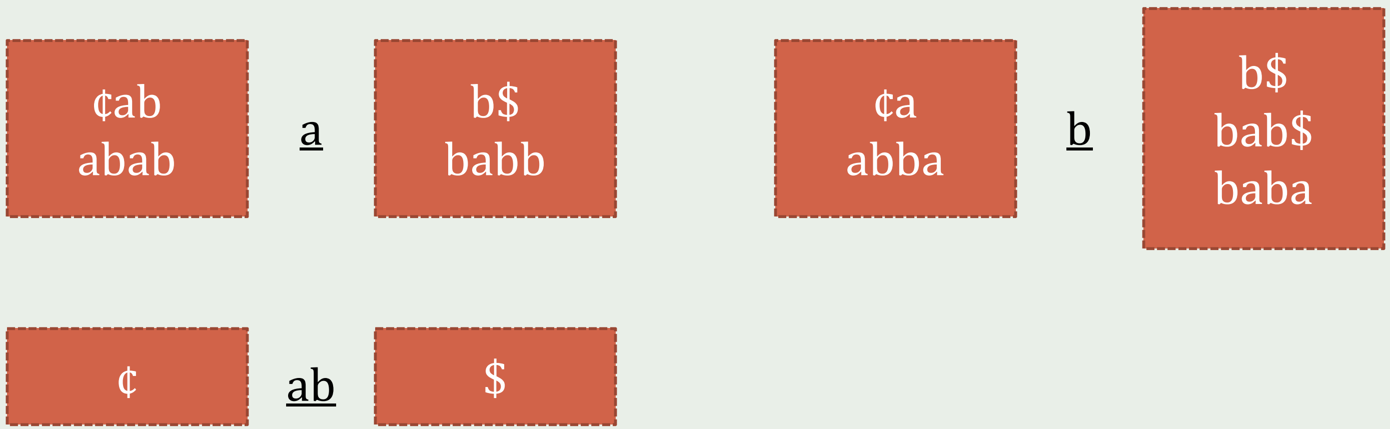
  $(\{\text{¢}ab, bab\}, \underline{a}, \{b\$, bab\}), (\{\text{¢}a, bba\}, \underline{b}, \{b\$, bab\}), (\text{¢}, \underline{ab}, \$)$

  And again we get:

  $ab\underline{a}bab \vdash_M aba\underline{b}b \vdash_M ab\underline{a}b \vdash_M a\underline{b}b \vdash_M \underline{ab} \vdash_M \lambda.$

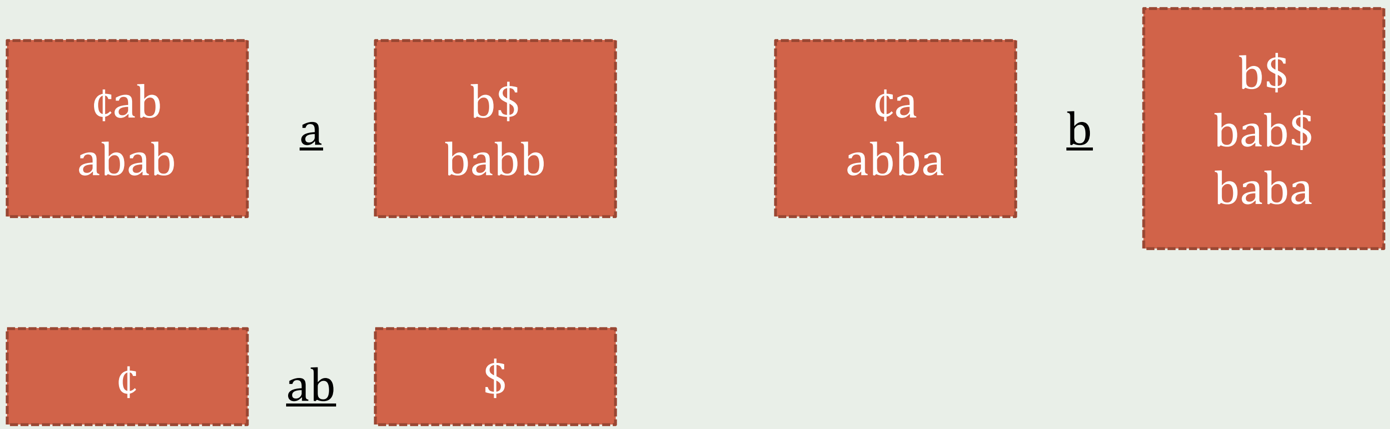
Learning Non-Context-Free Language

- Finally, for *k = 4* we get the required *4-cl-RA*-automaton *M*.



Learning Non-Context-Free Language

- Finally, for $k = 4$ we get the required *4-cl-RA*-automaton *M*.



- For this *4-cl-RA*-automaton *M* it can be shown, that:
$L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2^m} \mid m\geq 0\}$.

Conclusion

- We have seen that knowing some sample computations (or even reductions) of a *cl-RA*-automaton (or *Δcl-RA*-automaton) it is extremely simple to infer its instructions.

Conclusion

- We have seen that knowing some sample computations (or even reductions) of a *cl-RA*-automaton (or *Δcl-RA*-automaton) it is extremely simple to infer its instructions.
- The instructions of a *Δcl-RA*-automaton are human readable which is an advantage for their possible applications e.g. in linguistics.

Conclusion

- We have seen that knowing some sample computations (or even reductions) of a *cl-RA*-automaton (or *Δcl-RA*-automaton) it is extremely simple to infer its instructions.
- The instructions of a *Δcl-RA*-automaton are human readable which is an advantage for their possible applications e.g. in linguistics.
- Unfortunately, we still do not know whether *Δcl-RA*-automata can recognize all context-free languages.

Conclusion

- If we generalize *Δcl-RA*-automata by enabling them to use any number of auxiliary symbols: $\Delta_1, \Delta_2, \ldots, \Delta_n$ instead of single Δ, we will increase their power up-to context sensitive languages.

Conclusion

- If we generalize *Δcl-RA*-automata by enabling them to use any number of auxiliary symbols: $\Delta_1, \Delta_2, \ldots, \Delta_n$ instead of single Δ, we will increase their power up-to context sensitive languages.
  - Such automata can easily accept all languages generated by context-sensitive grammars with productions in *one-sided normal form*: $A \rightarrow a, A \rightarrow BC, AB \rightarrow AC$

    where A, B, C are nonterminals and a is a terminal.

Conclusion

- If we generalize *Δcl-RA*-automata by enabling them to use any number of auxiliary symbols: $\Delta_1, \Delta_2, \ldots, \Delta_n$ instead of single Δ, we will increase their power up-to context sensitive languages.
  - Such automata can easily accept all languages generated by context-sensitive grammars with productions in *one-sided normal form*: $A \rightarrow a$, $A \rightarrow BC$, $AB \rightarrow AC$

    where A, B, C are nonterminals and a is a terminal.
  - Penttonen showed that for every context-sensitive grammar there exists an equivalent grammar in one-sided normal form.

Open Problems

- What is the difference between language classes of $\mathcal{L}(k\text{-}cl\text{-}RA)$ and $\mathcal{L}(k\text{-}\Delta cl\text{-}RA)$ for different values of k?

Open Problems

- What is the difference between language classes of *ℒ(k-cl-RA)* and *ℒ(k-Δcl-RA)* for different values of k?
- Can *Δcl-RA*-automata recognize all string languages defined by ALD's?

Open Problems

- What is the difference between language classes of $\mathcal{L}(k\text{-}cl\text{-}RA)$ and $\mathcal{L}(k\text{-}\Delta cl\text{-}RA)$ for different values of k?
- Can $\Delta cl\text{-}RA$-automata recognize all string languages defined by ALD's?
- What is the relation between $\mathcal{L}(\Delta cl\text{-}RA)$ and the class of one counter languages, simple context-sensitive grammars (they have single nonterminal), etc?

References

- ČERNO, P., MRÁZ, F., Clearing restarting automata, tech. report., Department of Computer Science, Charles University, Prague, 2009.
- CHERUBINI, A., REGHIZZI, S.C., PIETRO, P.S., Associative language descriptions, Theoretical Computer Science, 270 (2002), 463-491.
- GREIBACH, S. A., The hardest context-free language, SIAM Journal on Computing, 2(4) (1973), 304-310.
- JANČAR, P., MRÁZ, F., PLÁTEK, M., VOGEL, J., Restarting automata, in: H. Reichel (Ed.), FCT'95, LNCS, Vol. 965, Springer, Berlin, 1995, 283-292.
- JANČAR, P., MRÁZ, F., PLÁTEK, M., VOGEL, J., On restarting automata with rewriting, in: Gh. Paun, A. Salomaa (Eds.), New Trends in Formal Language Theory (Control, Cooperation and Combinatorics), LNCS, Vol. 1218, Springer, Berlin, 1997, 119-136.
- JANČAR, P., MRÁZ, F., PLÁTEK, M., VOGEL, J., On monotonic automata with a restart operation, Journal of Automata, Languages and Combinatorics, 4(4) (1999), 287-311.
- LOPATKOVÁ, M., PLÁTEK, M., KUBOŇ, V., Modeling syntax of free word-order languages: Dependency analysis by reduction, in: V. Matoušek, P. Mautner, T. Pavelka (Eds.), Text, Speech and Dialogue: 8th International Conference, TSD 2005, LNCS, Vol. 3658, Springer, Berlin, 2005, 140-147.
- MATEESCU, A., SALOMAA, A., Aspects of classical language theory, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, volume 1 - Word, Language, Grammar, chapter 4, Springer, Berlin, 1997, 175-251.
- MRÁZ, F., OTTO, F., PLÁTEK, M., Learning analysis by reduction from positive data, in: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Proceedings ICGI 2006, LNCS, Vol. 4201, Springer, Berlin, 2006, 125-136.
- OTTO, F., Restarting automata and their relation to the chomsky hierarchy. In Z. Ésik, Z. Fülöp (Eds.), Developments in Language Theory, 7th International Conference, DLT 2003, Szeged, Hungary, LNCS, Vol. 2710, Springer, Berlin, 2003, 55-74.

WEB

http://www.petercerno.wz.cz/ra.html