Clearing Restarting Automata

PETER ČERNO
FRANTIŠEK MRÁZ
We propose a new restricted version of restarting automata called Clearing Restarting Automata.
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 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:
  - $\Sigma$ 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:

- $\Sigma$ is a finite nonempty alphabet, $\emptyset, \$ \notin \Sigma$.
- $I$ is a finite set of instructions $(x, z, y), x \in LC_k, y \in RC_k, z \in \Sigma^+$,
  - left context $LC_k = \Sigma^k \cup \emptyset, \Sigma^{\leq k-1}$
  - 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:

- $\Sigma$ is a finite nonempty alphabet, $\varnothing, \$ \not\in \Sigma$.
- $I$ is a finite set of instructions $(x, z, y), x \in LC_k, y \in RC_k, z \in \Sigma^+$,
  - left context $LC_k = \Sigma^k \cup \varnothing. \Sigma^{\leq k-1}$
  - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$ 

- The special symbols: $\varnothing$ 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:
  - $\Sigma$ is a finite nonempty alphabet, $\varnothing, \$ \not\in \Sigma$.
  - $I$ is a finite set of instructions $(x, z, y)$, $x \in LC_k$, $y \in RC_k$, $z \in \Sigma^+$,
    - left context $LC_k = \Sigma^k \cup \varnothing.\Sigma^{\leq k-1}$
    - right context $RC_k = \Sigma^k \cup \Sigma^{\leq k-1}.\$ 
  - The special symbols: $\varnothing$ and $\$ are called sentinels.
  - The width of the instruction $i = (x, z, y)$ is $|i| = |xzy|$.
A word $w = uzv$ can be \textit{rewritten} to $uv (u \sqsubseteq v \vdash_M uv)$ if and only if there exist an instruction $i = (x, z, y) \in I$ such that:

- $x \sqsupseteq \%u$ (\textit{x} is a suffix of \%u)
- $y \sqsubseteq v$ (\textit{y} is a prefix of \v)
A word \( w = uzv \) can be \textit{rewritten} to \( uv \) (\( u\mathrel{\not\sqsubseteq} v \vdash_M uv \)) if and only if there exist an instruction \( i = (x, z, y) \in I \) such that:

- \( x \sqsupseteq \mathcal{C}.u \) (\( x \) is a suffix of \( \mathcal{C}.u \))
- \( y \sqsubseteq v.\$ \) (\( y \) is a prefix of \( v.\$ \))

A word \( w \) is \textit{accepted} if and only if \( w \vdash_M^* \lambda \) where \( \vdash_M^* \) is reflexive and transitive closure of \( \vdash_M \).
A word $w = uzv$ can be rewritten to $uv$ ($u \preceq v \vdash_M uv$) if and only if there exist an instruction $i = (x, z, y) \in I$ such that:
- $x \supseteq \cdot u$ ($x$ is a suffix of $\cdot u$)
- $y \subseteq v.$ ($y$ 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 $L(M) = \{w \in \Sigma^* | M$ accepts $w\}$. 
Definition

- By \textit{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 \textit{cl-RA} we will denote the \textbf{class of all clearing restarting automata}.
- \(\mathcal{L}(k\text{-cl-RA})\) denotes the \textbf{class of all languages} accepted by \textit{k-cl-RA-automata}.
- Similarly \(\mathcal{L}(cl\text{-RA})\) denotes the \textbf{class of all languages} accepted by \textit{cl-RA-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.
- Similarly \( \mathcal{L}(cl-RA) \) denotes the class of all languages accepted by \( cl-RA \)-automata.
- \( \mathcal{L}(cl-RA) = \bigcup_{k \geq 1} \mathcal{L}(k-cl-RA) \).
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.
- Similarly \( \mathcal{L}(cl-RA) \) denotes the class of all languages accepted by \( cl-RA \)-automata.
- \( \mathcal{L}(cl-RA) = \bigcup_{k \geq 1} \mathcal{L}(k-cl-RA) \).
- **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](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](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-Regghizzi, Matteo Pradella, Pierluigi San Pietro.
  - See: [http://home.dei.polimi.it/sanpietr/ALD/ALD.html](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:
  - \( R1 = (a, ab, b) \)
  - \( R2 = (\$, ab, \$) \)
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:
  - \( R1 = (a, ab, b) \)
  - \( R2 = (\$, ab, \$) \)
- For instance:
  - \( aaaabbbb \xrightarrow{R1} aaabbb \)
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:
  - \( R1 = (a, ab, b) \)
  - \( R2 = (\$, ab, \$) \)
- For instance:
  - \( aaaabbbb \xleftarrow{R1} aaabbb \xleftarrow{R1} aabb \)
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:
  - $R1 = (a, ab, b)$
  - $R2 = (\$, ab, \$)$
- For instance:
  - $aaaabbbb \vdash R1 aaabbb \vdash R1 aabbb \vdash R1 ab$
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:
  - $R1 = (a, ab, b)$
  - $R2 = (\$, ab, \$)
- For instance:
  - $aaaabbbb \vdash^{R1} aaabbb \vdash^{R1} aabb \vdash^{R1} ab \vdash^{R2} \lambda$.
- 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 $\Sigma^*$. If $u \not\vdash^*_M v$ and $u \not\in L(M)$, then $v \not\in L(M)$.
  
  **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 $\Sigma^*$. 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, $)\}$. ■
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**: \( \mathcal{L}(k\text{-cl-RA}) \subset \mathcal{L}((k+1)\text{-cl-RA}) \), for all \( k \geq 1 \).
  - **Note**: The following language: \( \{ (c^kac^k)^n (c^kbc^k)^n \mid n \geq 0 \} \) belongs to \( \mathcal{L}((k+1)\text{-cl-RA}) - \mathcal{L}(k\text{-cl-RA}) \).

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

- **Theorem:** For each regular language $L \subseteq \Sigma^*$ there exist a $k\text{-cl-RA}$-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.
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.
  - We add corresponding instruction \( i_z = (\cdot, u, v, w) \).
Some Theorems

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

- **Theorem:** For each regular language \( L \subseteq \Sigma^* \) there exist a \( k\text{-cl-RA}-\)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 \( i_z = (\cdot, u, v, w) \).
  - For each accepted \( z \in \Sigma^{<n} - \{ \lambda \} \) we add instruction \( i_z = (\cdot, z, \$) \).
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 $L_1 = \{a^n b^n \mid n \geq 0\} \cup \{\lambda\}$ 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 $L_2 \cdot L_3$ are not recognized by any cl-RA-automaton.
Some Theorems

Let $L_2 = \{a^n b^n / n \geq 0\}$ and $L_3 = \{a^n b^{2n} / n \geq 0\}$ 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 $L_2 \cdot L_3$ 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 $\{a^n b^n / n \geq 0\} \cup \{c^n d^{2n} / n \geq 0\}$ 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:
  - $L_4 = \{a^n cb^n \mid n \geq 0\} \cup \{a^m b^m \mid m \geq 0\}$
  - $L_5 = \{a^n c b^m \mid n, m \geq 0\} \cup \{\lambda\}$
  - $L_6 = \{a^m b^m \mid m \geq 0\}$

can be recognized by a 1-cl-RA-automaton.
Some Theorems

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

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

- **Corollary:** $\mathcal{L}(cl-RA)$ 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.

- **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.
  - 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:
    - $({\emptyset}\cup\Sigma, (,), \Sigma\cup\{\}$), where $\Sigma = \{(,),\}$ and
    - $(A, w, B) = \{(a, w, b) \mid a\in A, b\in B\}$. 
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:
  - $({\emptyset}\cup\Sigma, (\ ), \Sigma\cup\{\$\})$, 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 $\Sigma = \{\alpha, +, *, (, )\}$. 
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.
Suppose that we want to check correctness of arithmetic expressions over the alphabet $\Sigma = \{ \alpha, +, *, (, ) \}$.

For example $\alpha + (\alpha \ast \alpha + \alpha)$ is correct, $\alpha \ast + \alpha$ is not.

The priority of the operations is considered.
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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha\alpha + ((\alpha + \alpha) + (\alpha + \alpha\alpha))\alpha$</td>
<td>($\epsilon, \alpha^*, \alpha$)</td>
</tr>
<tr>
<td>$\alpha + ((\alpha + \alpha) + (\alpha + \alpha\alpha))\alpha$</td>
<td>($\alpha, +\alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha + \alpha\alpha))\alpha$</td>
<td>($, *\alpha, $)</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha + \alpha\alpha))$</td>
<td>($+, \alpha^*, \alpha$)</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha + \alpha))$</td>
<td>($, \alpha+, \alpha$)</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha))$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha))$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + ((\alpha) + (\alpha))$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + ((\alpha))$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\alpha)$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\alpha)$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\alpha)$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\alpha)$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\alpha)$</td>
<td>($, \alpha, )$ )</td>
</tr>
<tr>
<td>$\alpha + (\epsilon)$</td>
<td>($, \alpha+, (\epsilon)$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>($, \alpha+, \epsilon$)</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>($, \alpha+, (\epsilon$)</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Nondeterminism

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

and the word: \( cbbabbb \).
Nondeterminism

- Assume the following instructions:
  - $R_1 = (bb, a, bbbb)$
  - $R_2 = (bb, bb, \$$)
  - $R_3 = (\$, cbb, \$$)

and the word: $cbbabbbb$. Then:
  - $cbbabbbb \vdash_{R_1} cbbbbb \vdash_{R_2} cbbb \vdash_{R_2} cbb \vdash_{R_3} \lambda$. 
Nondeterminism

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

and the word: $cbbabbb$. Then:
  - $cbbabbb \vdash^{R1} cbbabbb \vdash^{R2} cbbabbb \vdash^{R2} cbb \vdash^{R3} \lambda$.

- **But** if we have started with $R2$:
  - $cbbabbb \vdash^{R2} cbbabbb$

  then it would not be possible to continue.
Nondeterminism

- Assume the following instructions:
  - $R_1 = (bb, a, bbbb)$
  - $R_2 = (bb, bb, \$)$
  - $R_3 = (\$, cbb, \$)$

and the word: $cbbabbb$. Then:

- $cbbabbb \vdash_{R_1} cbbbbbb \vdash_{R_2} cbbbbb \vdash_{R_2} cbb \vdash_{R_3} \lambda$.

- **But** if we have started with $R_2$:
  - $cbbabbb \vdash_{R_2} cbbabb$

then it would not be possible to continue.

- $\Rightarrow$ 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.
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 $\varphi: L = \varphi^{-1}(H)$. 
By S. A. Greibach, definition from Section 10.5 of M. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.

Let $\Sigma = \{a_1, a_2, a_1, a_2, \#, c\}$, $d \not\in \Sigma$. 
Greibach’s Hardest CFL

- Let $\Sigma = \{a_1, a_2, a_1, a_2, \#, c\}$, $d \notin \Sigma$.
- Let $D_2$ be *Semi-Dyck language* on $\{a_1, a_2, a_1, a_2\}$ generated by the grammar: $S \rightarrow \lambda / SS / a_1Sa_1 / a_2Sa_2$. 
Greibach’s Hardest CFL

- Let $\Sigma = \{a_1, a_2, a_1', a_2', #, c\}$, $d \notin \Sigma$.
- Let $D_2$ be Semi-Dyck language on $\{a_1, a_2, a_1', a_2'\}$ generated by the grammar: $S \rightarrow \lambda / SS / a_1Sa_1 / a_2Sa_2$.
- Then $H = \{\lambda\} \cup \{\prod_{i=1}^{n} x_i y_i c y_i c z_i d \mid n \geq 1, y_1 y_2 ... y_n \in \#D_2, x_i, z_i \in \Sigma^*\}$,
  - $y_1 \in \# . \{a_1', a_2', a_1, a_2\}^*$,
  - $y_i \in \{a_1', a_2', a_1, a_2\}^*$ 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)*
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.
- $k$-$\Delta$-clearing restarting automaton ($k$-$\Delta cl$-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:
Let $k$ be a positive integer.

$k$-$\Delta$-clearing restarting automaton ($k$-$\Delta$cl-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:

- $\Sigma$ is a finite nonempty alphabet, $\$ \not \in \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$.
Let $k$ be a positive integer.

$k$-$\Delta$-clearing restarting automaton ($k$-$\Delta$cl-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:

- $\Sigma$ is a finite nonempty alphabet, $\mathcal{C}, \$, $\Delta \notin \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$. 
- $I$ is a finite set of instructions of the following forms:
  - (1) $(x, z \rightarrow \lambda, y)$
  - (2) $(x, z \rightarrow \Delta, y)$
Let $k$ be a positive integer.

$k$-$\Delta$-clearing restarting automaton ($k$-$\Delta$cl-RA-automaton for short) is a couple $M = (\Sigma, I)$, where:

- $\Sigma$ is a finite nonempty alphabet, $\emptyset, \$, $\Delta \not\in \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$.
- $I$ is a finite set of instructions of the following forms:
  - (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 $LC_k = \Gamma^k \cup \emptyset, \Gamma^{\leq k-1}$
  - right context $RC_k = \Gamma^k \cup \Gamma^{\leq k-1}, \$
A word $w = uzv$ can be \textit{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:

- $x \sqsupseteq \cdot u$ (x is a suffix of $\cdot u$)
- $y \sqsubseteq v.$ (y is a prefix of $v.$)
A word $w = uzv$ can be rewritten to $usv$ ($u \sqsubseteq v \vdash_M usv$) if and only if there exist an instruction $i = (x, z \rightarrow s, y) \in I$ such that:
- $x \sqsubseteq \epsilon$.u ($x$ is a suffix of $\epsilon$.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 \( usv \) \((u \not\subseteq v \vdash_M usv)\) if and only if there exist an instruction \( i = (x, z \rightarrow s, y) \in I \) such that:

- \( x \supseteq \_u \) (\( x \) is a suffix of \( \_u \))
- \( y \subseteq v.\$ \) (\( y \) 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 \( k\)-\( \Delta \text{cl-RA-automaton} \) \( M \) **recognizes** the language \( L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \} \).
By $\Delta cl$-$RA$ we will denote the class of all $\Delta$-clearing restarting automata.
By $\Delta cl$-$RA$ we will denote the class of all $\Delta$-clearing restarting automata.

$L(k\cdot\Delta cl$-$RA)$ denotes the class of all languages accepted by $k$-$\Delta cl$-$RA$-automata.
Definition

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

By $\Delta cl$-$RA$ we will denote the class of all $\Delta$-clearing restarting automata.

$\mathcal{L}(k$-$\Delta cl$-$RA)$ denotes the class of all languages accepted by $k$-$\Delta cl$-$RA$-automata.

Similarly $\mathcal{L}(\Delta cl$-$RA)$ denotes the class of all languages accepted by $\Delta cl$-$RA$-automata.

$\mathcal{L}(\Delta cl$-$RA) = \bigcup_{k \geq 1} \mathcal{L}(k$-$\Delta cl$-$RA)$.
By \(\Delta cl-RA\) we will denote the class of all \(\Delta\)-clearing restarting automata.

\(\mathcal{L}(k-\Delta cl-RA)\) denotes the class of all languages accepted by \(k-\Delta cl-RA\)-automata.

Similarly \(\mathcal{L}(\Delta cl-RA)\) denotes the class of all languages accepted by \(\Delta cl-RA\)-automata.

\(\mathcal{L}(\Delta cl-RA) = \bigcup_{k \geq 1} \mathcal{L}(k-\Delta cl-RA)\).

**Note:** For every \(\Delta cl-RA\) \(M\): \(\lambda \vdash^*_M \lambda\) hence \(\lambda \in L(M)\). If we say that \(\Delta cl-RA\) \(M\) 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:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (\emptyset, c \rightarrow \lambda, \$$)$
  - $R\Delta1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta2 = (\emptyset, a\Delta b \rightarrow \lambda, \$$)$
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:
  - $Rc1 = (a, c \rightarrow \Delta, b)$, $Rc2 = (\emptyset, c \rightarrow \lambda, \$$)
  - $R\Delta1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta2 = (\emptyset, a\Delta b \rightarrow \lambda, \$$)
- For instance:
  - $aaa_c bbb \xrightarrow{Rc1} aa\Delta bb$
Example

- **Language** $L = \{a^n cb^n \mid n \geq 0\}$.
- Can be recognized by the $1$-$\Delta$cl-$RA$ $M = (\{a, b, c\}, I)$, where the instructions $I$ are:
  - $Rc_1 = (a, c \rightarrow \Delta, b)$, $Rc_2 = (\emptyset, c \rightarrow \lambda, \$$)
  - $R\Delta_1 = (a, a\Delta b \rightarrow \Delta, b)$, $R\Delta_2 = (\emptyset, a\Delta b \rightarrow \lambda, \$$)
- **For instance:**
  - $aaacbbb \xrightarrow{Rc_1} aa\Delta bb \xrightarrow{R\Delta_1} a\Delta b$
Example

- Language $L = \{a^n cb^n \mid n \geq 0\}$.
- Can be recognized by the 1-Δcl-RA $M = (\{a, b, c\}, I)$, where the instructions $I$ are:
  - $Rc_1 = (a, c \rightarrow \Delta, b), Rc_2 = (\emptyset, c \rightarrow \lambda, \$)
  - $R\Delta_1 = (a, a\Delta b \rightarrow \Delta, b), R\Delta_2 = (\emptyset, a\Delta b \rightarrow \lambda, \$)
- For instance:
  - $aaacbbb \vdash^{Rc_1} aa\Delta bb \vdash^{R\Delta_1} a\Delta b \vdash^{R\Delta_2} \lambda$.
- 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 $1$-$\Delta cl$-$RA$-automaton.
Theorem: Greibach’s Hardest CFL $H$ is recognized by a $1$-$\Delta cl$-$RA$-automaton.

- **Idea.** Suppose that we have $w \in H$:
  
  $$ w = \epsilon x_1cy_1cz_1d \ x_2cy_2cz_2d \ldots x_ncy_ncz_nd $$
Theorem: Greibach’s Hardest CFL \( H \) is recognized by a 1-\( \Delta \)-cl-RA-automaton.

- **Idea.** Suppose that we have \( w \in H \):
  \[
  w = \epsilon x_1 c y_1 c z_1 d \ x_2 cy_2 cz_2 d \ldots x_n cy_n cz_n d 
  \]

- In the first phase we start with deleting letters \( ( \) from the right side of \( \epsilon \) and from the left and right sides of the letters \( d \).
Back to Greibach’s Hardest CFL

- **Theorem:** Greibach’s Hardest CFL $H$ is recognized by a 1-$\Delta$cl-RA-automaton.

  - **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 $\Sigma = \{a_1, a_2, a_1, a_2, \#, c\}$ ) 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:
    $$\$ c y_1 c d \ c y_2 c d \ldots c y_n c d \$$, we introduce the $\Delta$ symbols:
    $$\$ \Delta y_1 \Delta y_2 \Delta \ldots \Delta y_n \Delta \$$
Back to Greibach’s Hardest CFL

**Theorem:** Greibach’s Hardest CFL $H$ is recognized by a $1$-$\Delta$cl-RA-automaton.

- **Idea.** Suppose that we have $w \in H$:
  
  $$w = \phi 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 $\Sigma = \{a_1, a_2, a_1', a_2', #, c\}$ ) from the right side of $\phi$ and from the left and right sides of the letters $d$.

- As soon as we think that we have the following word:
  
  $$\phi cy_1cd \ cy_2cd \ldots cy_ncd \$$

  we introduce the $\Delta$ symbols:

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

<table>
<thead>
<tr>
<th>Instructions for the first phase:</th>
<th>Instructions for the second phase:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) ($\emptyset$, $\Sigma \rightarrow \lambda$, $\Sigma$)</td>
<td>(7) ($\Gamma, a_1a_1 \rightarrow \lambda$, $\Gamma - {#}$)</td>
</tr>
<tr>
<td>(2) ($\Sigma$, $\Sigma \rightarrow \lambda$, $d$)</td>
<td>(8) ($\Gamma, a_2a_2 \rightarrow \lambda$, $\Gamma - {#}$)</td>
</tr>
<tr>
<td>(3) ($d$, $\Sigma \rightarrow \lambda$, $\Sigma$)</td>
<td>(9) ($\Gamma, a_1\Delta a_1 \rightarrow \Delta$, $\Gamma - {#}$)</td>
</tr>
<tr>
<td>(4) ($\emptyset$, $c \rightarrow \Delta$, $\Sigma \cup {\Delta}$)</td>
<td>(10) ($\Gamma, a_2\Delta a_2 \rightarrow \Delta$, $\Gamma - {#}$)</td>
</tr>
<tr>
<td>(5) ($\Sigma \cup {\Delta}$, $cdc \rightarrow \Delta$, $\Sigma \cup {\Delta}$)</td>
<td>(11) ($\Sigma - {c}$, $\Delta \rightarrow \lambda$, $\Delta$)</td>
</tr>
<tr>
<td>(6) ($\Sigma \cup {\Delta}$, $cd \rightarrow \Delta$, $$)</td>
<td>(12) ($\emptyset$, $\Delta#\Delta \rightarrow \lambda$, $$)</td>
</tr>
</tbody>
</table>
Instructions recognizing Hardest CFL \( H \)

- Suppose \( \Sigma = \{a_1, a_2, a_1 a_2, \#, c\}, d \notin \Sigma, \Gamma = \Sigma \cup \{d, \Delta\} \).

<table>
<thead>
<tr>
<th>Instructions for the first phase:</th>
<th>Instructions for the second phase:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) (( \emptyset, \Sigma \rightarrow \lambda, \Sigma ))</td>
<td>(7) (( \Gamma, a_1 a_1 \rightarrow \lambda, \Gamma - {#} ))</td>
</tr>
<tr>
<td>(2) (( \Sigma, \Sigma \rightarrow \lambda, d ))</td>
<td>(8) (( \Gamma, a_2 a_2 \rightarrow \lambda, \Gamma - {#} ))</td>
</tr>
<tr>
<td>(3) (d, ( \Sigma \rightarrow \lambda, \Sigma ))</td>
<td>(9) (( \Gamma, a_1 \Delta a_1 \rightarrow \Delta, \Gamma - {#} ))</td>
</tr>
<tr>
<td>(4) (( \emptyset, c \rightarrow \Delta, \Sigma \cup {\Delta} ))</td>
<td>(10) (( \Gamma, a_2 \Delta a_2 \rightarrow \Delta, \Gamma - {#} ))</td>
</tr>
<tr>
<td>(5) (( \Sigma \cup {\Delta}, cdc \rightarrow \Delta, \Sigma \cup {\Delta} ))</td>
<td>(11) (( \Sigma - {c}, \Delta \rightarrow \lambda, \Delta ))</td>
</tr>
<tr>
<td>(6) (( \Sigma \cup {\Delta}, cd \rightarrow \Delta, $ ))</td>
<td>(12) (( \emptyset, \Delta # \Delta \rightarrow \lambda, $ ))</td>
</tr>
</tbody>
</table>

- In fact, there is no such thing as a \textit{first phase} or a \textit{second phase}. We have only instructions.
• Suppose $\Sigma = \{a_1, a_2, a_1 a_2, \#, c\}$, $d \notin \Sigma$, $\Gamma = \Sigma \cup \{d, \Delta\}$.

<table>
<thead>
<tr>
<th>Instructions for the first phase:</th>
<th>Instructions for the second phase:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) ($\emptyset, \Sigma \to \lambda, \Sigma$)</td>
<td>(7) ($\Gamma, a_1 a_1 \to \lambda, \Gamma - {#}$)</td>
</tr>
<tr>
<td>(2) ($\Sigma, \Sigma \to \lambda, d$)</td>
<td>(8) ($\Gamma, a_2 a_2 \to \lambda, \Gamma - {#}$)</td>
</tr>
<tr>
<td>(3) ($d, \Sigma \to \lambda, \Sigma$)</td>
<td>(9) ($\Gamma, a_1 \Delta a_1 \to \Delta, \Gamma - {#}$)</td>
</tr>
<tr>
<td>(4) ($\emptyset, c \to \Delta, \Sigma \cup {\Delta}$)</td>
<td>(10) ($\Gamma, a_2 \Delta a_2 \to \Delta, \Gamma - {#}$)</td>
</tr>
<tr>
<td>(5) ($\Sigma \cup {\Delta}, cdc \to \Delta, \Sigma \cup {\Delta}$)</td>
<td>(11) ($\Sigma - {c}, \Delta \to \lambda, \Delta$)</td>
</tr>
<tr>
<td>(6) ($\Sigma \cup {\Delta}, cd \to \Delta, $$)</td>
<td>(12) ($\emptyset, \Delta # \Delta \to \lambda, $$)</td>
</tr>
</tbody>
</table>

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

• Theorem: $H \subseteq L(M)$, $H \supseteq L(M)$. 
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 $u_i \vdash_M v_i, i = 1, 2, ..., n$ 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 \( k\text{-}cl\text{-}RA\)-automaton \( M = (\Sigma, I) \), where 
\[
I = \{ ( \text{Suff}_k(\$, x_i), z_i, \text{Pref}_k(y_i, \$) ) \mid i = 1, \ldots, n \}. 
\]
Step 3: Construct a $k$-cl-RA-automaton $M = (\Sigma, I)$, where $I = \{ (\text{Suff}_k(\$x_i), z_i, \text{Pref}_k(y_i,\$)) | i = 1, ..., n \}$. 

- $\text{Pref}_k(u)$ ($\text{Suff}_k(u)$, 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 3: Construct a $k$-cl-RA-automaton $M = (\Sigma, I)$, where $I = \{ (\text{Suff}_k(\$.x_i), z_i, \text{Pref}_k(y_i,\$) ) \mid i = 1, \ldots, n \}$.  

- $\text{Pref}_k(u)$ ($\text{Suff}_k(u)$, 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

Step 3: Construct a $k$-cl-RA-automaton $M = (\Sigma, I)$, where $I = \{ (\text{Suff}_k(\$x_i), z_i, \text{Pref}_k(y_i,\$)) | i = 1, \ldots, n \}$.

- $\text{Pref}_k(u)$ ($\text{Suff}_k(u)$, 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.

Step 5: 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.
Even if the algorithm is very simple, it can be used to infer some non-trivial clearing (and after some generalization also $\Delta$-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**: \( \mathcal{L}(Δcl-RA) \subseteq CSL \), where CSL denotes the class of context-sensitive languages.
Theorem: There exists a \textit{k-cl-RA-automaton} $M$ recognizing a language that is \textit{not} context-free.
Theorem: There exists a \textit{k-cl-RA}-automaton $M$ recognizing a language that is not context-free.

\textbf{Idea.} We try to create a \textit{k-cl-RA}-automaton $M$ such that $L(M) \cap \{(ab)^n \mid n > 0\} = \{(ab)^{2m} \mid m \geq 0\}$.
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 $L(M) \cap \{(ab)^n / n > 0\} = \{(ab)^{2m} / m \geq 0\}$.
  - 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:**
  \$abababababababababababababababababababab\$
Learning Non-Context-Free Language

- **Example:**
  \[ \L\{ abababababababab \} \not\vdash_M \L\{ abababababababa \} \]
  \[ \L\{ ababababababababbb \} \vdash_M \L\{ abababababababa \} \]
  \[ \L\{ ababababababababb \} \not\vdash_M \L\{ ababababababababbb \} \]
  \[ \L\{ ababababababababb \} \vdash_M \L\{ ababababababababbc \} \]
  \[ \L\{ ababababababababb \} \not\vdash_M \L\{ abababababababbb \} \]
  \[ \L\{ abababababababbb \} \vdash_M \L\{ abababababababbb \} \]
Learning Non-Context-Free Language

- **Example:**
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababb} \vdash_M \]
  
  \[ \cdot \text{abababababababb} \vdash_M \cdot \text{abababababababb} \vdash_M \]
  
  \[ \cdot \text{abababababababb} \vdash_M \cdot \text{ababababababababb} \vdash_M \]
  
  \[ \cdot \text{abababababababb} \vdash_M \cdot \text{ababababababababb} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
  
  \[ \cdot \text{abababababababab} \vdash_M \cdot \text{abababababababab} \vdash_M \]
Learning Non-Context-Free Language

Example:

$\frac{ababababababab}{M} \frac{ababababababab}{M}$

$\frac{abababababbab}{M} \frac{abababababbbabb}{M}$

$\frac{abbabbababb}{M} \frac{ababababbbabb}{M}$

$\frac{abbbababab}{M} \frac{ababababab}{M}$

$\frac{abababab}{M} \frac{ababab}{M}$

$\frac{abab}{M}$

$\frac{abbabb}{M}$
Learning Non-Context-Free Language

- **Example:**

  \[¢ \text{ababababababababab} \vdash M \vdash ¢ \text{ababababababababb} \vdash M\]
  \[¢ \text{abababababababb} \vdash M \vdash ¢ \text{abababababababb} \vdash M\]
  \[¢ \text{abababababababb} \vdash M \vdash ¢ \text{ababababababababb} \vdash M\]
  \[¢ \text{abababababababb} \vdash M \vdash ¢ \text{ababababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
  \[¢ \text{abababababababab} \vdash M \vdash ¢ \text{abababababababab} \vdash M\]
Learning Non-Context-Free Language

Example:

\[ \& \mathit{ababababababab} \vdash M \& \mathit{ababababababab\ babb} \vdash M \]
\[ \& \mathit{ababababbbabb} \vdash M \& \mathit{ababbbabbbabb} \vdash M \]
\[ \& \mathit{abbbabbbabbb} \vdash M \& \mathit{ababbabbbabbb} \vdash M \]
\[ \& \mathit{ababbabab} \vdash M \& \mathit{abab} \vdash M \]
\[ \& \mathit{abab} \vdash M \& \mathit{ab} \vdash M \& \mathit{ab} \vdash M \& \mathit{\lambda} \vdash M \& \mathit{accept} \]
Learning Non-Context-Free Language

- **Example:**
  \[ \varepsilon \text{ababababababab} \vdash_M \varepsilon \text{abababababababb} \]
  \[ \varepsilon \text{ababababababb} \vdash_M \varepsilon \text{abababbabbabb} \]
  \[ \varepsilon \text{abababbabbabb} \vdash_M \varepsilon \text{abababbabbab} \]
  \[ \varepsilon \text{abababbabab} \vdash_M \varepsilon \text{abababab} \]
  \[ \varepsilon \text{abababab} \vdash_M \varepsilon \text{ababab} \]
  \[ \varepsilon \text{ababab} \vdash_M \varepsilon \text{abab} \]
  \[ \varepsilon \text{abab} \vdash_M \varepsilon \text{ab} \]
  \[ \varepsilon \text{ab} \vdash_M \varepsilon \lambda \]

  From this sample computation we can collect 15 reductions with unambiguous factorizations and use them as an input to our algorithm.
The only variable we have to choose is $k$ - the length of the context of the instructions.
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:
  
  $$(b, a, b), (a, b, b), (\$, ab, \$$)$$

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

  $ababab \vdash_M ababb \vdash_M aabbb \vdash_M abbb \vdash_M ab \vdash_M \lambda$. 


Learning Non-Context-Free Language

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

  $(ab, a, \{b$, ba$\}), (\{\$, ba\}, b, \{b$, ba$\}), (\$, ab, \$)$
For $k = 2$ we get the following set of instructions:

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

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

$ababab \vdash_M abab \vdash_M ab \vdash_M b \vdash_M b \vdash_M aab \vdash_M ab \vdash_M \lambda.$
Learning Non-Context-Free Language

- For $k = 2$ we get the following set of instructions:
  \[(ab, a, \{b$, ba\}), ({\$a, ba}, b, \{b$, ba\}), (\$, ab, \$)\]
  But then the automaton would accept the word $ababab$ which does not belong to $L$:
  \[ababab \vdash_M abab \vdash_M abab \vdash_M ab \vdash_M ab \vdash_M \ldots\]

- For $k = 3$ we get the following set of instructions:
  \[{\$ab, bab}, a, \{b$, bab\}), ({\$a, bba}, b, \{b$, bab\}), (\$, ab, \$)\]
Learning Non-Context-Free Language

- For $k = 2$ we get the following set of instructions:
  $$(ab, a, \{b$, $ba\}), (\{\$, ba\}, b, \{b$, $ba\}), (\$, ab, \$$)$$

  But then the automaton would accept the word $ababab$ which does not belong to $L$:
  $$ababab \xrightarrow{\text{M}} abab \xrightarrow{\text{M}} abb \xrightarrow{\text{M}} ab \xrightarrow{\text{M}} \lambda.$$ 

- For $k = 3$ we get the following set of instructions:
  $$(\{\$, ab\}, a, \{b$, bab\}), (\{\$, bba\}, b, \{b$, bab\}), (\$, ab, \$$)$$

  And again we get:
  $$ababab \xrightarrow{\text{M}} abab \xrightarrow{\text{M}} abab \xrightarrow{\text{M}} abb \xrightarrow{\text{M}} ab \xrightarrow{\text{M}} \lambda.$$
Finally, for \( k = 4 \) we get the required 4-cl-RA-automaton \( M \).
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:

$$L(M) \cap \{(ab)^n \mid n>0\} = \{(ab)^{2m} \mid m \geq 0\}.$$
Conclusion

- We have seen that knowing some sample computations (or even reductions) of a $cl$-$RA$-automaton (or $\Delta cl$-$RA$-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 $\Delta cl$-RA-automaton) it is extremely simple to infer its instructions.
- The instructions of a $\Delta cl$-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 $\Delta cl$-RA-automaton) it is extremely simple to infer its instructions.
- The instructions of a $\Delta cl$-RA-automaton are human readable which is an advantage for their possible applications e.g. in linguistics.
- Unfortunately, we still do not know whether $\Delta cl$-RA-automata can recognize all context-free languages.
If we generalize $\Delta cl$-RA-automata by enabling them to use any number of auxiliary symbols: $\Delta_1, \Delta_2, ..., \Delta_n$ instead of single $\Delta$, we will increase their power up-to context sensitive languages.
Conclusion

- If we generalize $\Delta_{cl-RA}$-automata by enabling them to use any number of auxiliary symbols: $\Delta_1, \Delta_2, \ldots, \Delta_n$ instead of single $\Delta$, 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.
If we generalize \( \Delta cl\text{-}RA\text{-}automata \) by enabling them to use any number of auxiliary symbols: \( \Delta_1, \Delta_2, \ldots, \Delta_n \) instead of single \( \Delta \), 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 $\mathcal{L}(k-cl-RA)$ and $\mathcal{L}(k-\Delta cl-RA)$ for different values of $k$?
Open Problems

- What is the difference between language classes of $\mathcal{L}(k-cl-RA)$ and $\mathcal{L}(k-\Delta cl-RA)$ for different values of $k$?
- Can $\Delta cl-RA$-automata recognize all string languages defined by ALD’s?
Open Problems

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