

Characterising REGEX Languages by Regular Languages Equipped with Factor-Referencing

Markus L. Schmid

Trier University, Germany

DLT 2014

The Problems of Context-Freeness

- “The world” is not context-free.

The Problems of Context-Freeness

- “The world” is **not context-free**.
- **Context-sensitive** languages are often too powerful (vast expressive power, parsing intractable, undecidability, etc.)

The Problems of Context-Freeness

- “The world” is **not context-free**.
- **Context-sensitive** languages are often too powerful (vast expressive power, parsing intractable, undecidability, etc.)
- We often need language classes with some “**non-context-free features**”, while at the same time **weaker than context-sensitive**.

The Problems of Context-Freeness

- “The world” is **not context-free**.
- **Context-sensitive** languages are often too powerful (vast expressive power, parsing intractable, undecidability, etc.)
- We often need language classes with some “**non-context-free features**”, while at the same time **weaker than context-sensitive**.

regulated rewriting (add control mechanisms to context-free grammars)

mildly context-sensitive (allow only a little bit of context-sensitivity).

Typical Non-Context-Free Features

Reduplication $\{ww \mid w \in \Sigma^*\}$

Multiple agreements $\{a^n b^n c^n \mid n \geq 1\}$

Crossed agreements $\{a^n b^m c^n d^m \mid n, m \geq 1\}$

We solely focus on reduplication.

Typical Non-Context-Free Features

Reduplication $\{ww \mid w \in \Sigma^*\}$

Multiple agreements $\{a^n b^n c^n \mid n \geq 1\}$

Crossed agreements $\{a^n b^m c^n d^m \mid n, m \geq 1\}$

We solely focus on **reduplication**.

Language descriptor/grammars models tailored to reduplication:

- L systems,
- pattern languages,
- H-systems,
- Wijngaarden grammars, macro grammars, Indian parallel grammars, deterministic iteration grammars,
- pattern expressions, synchronized regular expressions, EH-expressions, extended regular expressions with backreferences (REGEX).

Reference-Words - Idea

abacbcxcbya

The diagram illustrates the string "abacbcxcbya" with three blue curly braces above it. The first brace is under "bc" and labeled "y". The second brace is under "xc" and labeled "z". The third brace is under "ab" and labeled "x".

Reference-Words - Idea

a b a c b c x c b z y a

The diagram illustrates the decomposition of the string "abcabcxcbya" into three parts: "abc" (labeled x), "bcx" (labeled y), and "cbza" (labeled z). The letters are color-coded: 'a' is red, 'b' is green, 'c' is blue, 'x' is red, 'c' is green, 'b' is blue, 'z' is red, 'y' is green, and 'a' is blue.

Reference-Words - Idea

a b a c b c b a c c b z y a

The diagram illustrates the string "abcabcabcbyza" with three overlapping substrings marked by brackets and labeled with letters x, y, and z. Substring x (red) is "abc", y (blue) is "abc", and z (blue) is "abcbyza".

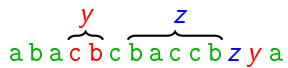
Reference-Words - Idea

a b a c b c b a c c b z y a

The diagram shows the string "abc bcbaccbzya" with two brackets above it. The first bracket is labeled "y" and spans the "bc" part. The second bracket is labeled "z" and spans the "baccb" part. The characters "z" and "y" are colored blue, while the others are green.

Reference-Words - Idea

a b a ^y c b c ^z b a c c b z y a

The diagram shows the string "a b a c b c b a c c b z y a". A red bracket is positioned above the characters 'c' and 'b' at indices 4 and 5, with a red 'y' above it. A blue bracket is positioned above the characters 'b', 'a', 'c', 'c', and 'b' at indices 7, 8, 9, 10, and 11, with a blue 'z' above it. The characters 'c' and 'b' at indices 4 and 5 are colored red, while the characters 'b', 'a', 'c', 'c', and 'b' at indices 7, 8, 9, 10, and 11 are colored blue. The other characters are black.

Reference-Words - Idea

a b a ^y c b c ^z b a c c b z c b a

The diagram shows a sequence of characters: a, b, a, c, b, c, b, a, c, c, b, z, c, b, a. Above the characters 'c', 'b', and 'c' (indices 4, 5, and 6) is a red bracket labeled 'y'. Above the characters 'b', 'a', 'c', 'c', and 'b' (indices 7, 8, 9, 10, and 11) is a blue bracket labeled 'z'. The characters 'a', 'b', 'a', 'c', 'b', 'c', 'z', 'c', 'b', 'a' are colored green, red, blue, green, red, blue, green, red, blue, green respectively.

Reference-Words - Idea

a b a c b c $\overbrace{b a c c b}^z$ z c b a

Reference-Words - Idea

a b a c b c ^z b a c c b z c b a

Reference-Words - Idea

a b a c b c $\overbrace{b a c c b}^z$ b a c c b c b a

Reference-Words - Idea

abacbcbaaccbbaccbcba

Reference-Words - Formal Definition

- Σ is a finite alphabet.

Reference-Words - Formal Definition

- Σ is a finite alphabet.
- $\Gamma = \{[x_i,]_{x_i}, x_i \mid i \in \mathbb{N}\}$.

Reference-Words - Formal Definition

- Σ is a finite alphabet.
- $\Gamma = \{[x_i,]_{x_i}, x_i \mid i \in \mathbb{N}\}$.
- $[x_i$ and $]_{x_i}$ are called **parentheses**, x_i is called **variable**.

Reference-Words - Formal Definition

- Σ is a finite alphabet.
- $\Gamma = \{[x_i,]_{x_i}, x_i \mid i \in \mathbb{N}\}$.
- $[x_i$ and $]_{x_i}$ are called **parentheses**, x_i is called **variable**.
- A **ref-word over Σ** is a word $w \in (\Sigma \cup \Gamma)^*$.

Reference-Words - Formal Definition

- Σ is a finite alphabet.
- $\Gamma = \{[x_i,]_{x_i}, x_i \mid i \in \mathbb{N}\}$.
- $[x_i$ and $]_{x_i}$ are called **parentheses**, x_i is called **variable**.
- A **ref-word over Σ** is a word $w \in (\Sigma \cup \Gamma)^*$.
- A ref-word is **valid** if, for every $i \in \mathbb{N}$,
 - ▶ only well-formed, non-nested pairs of parentheses $[x_i,]_{x_i}$,
 - ▶ no x_i inside of $[x_i \dots]_{x_i}$.

Reference-Words - Formal Definition

- Σ is a finite alphabet.
- $\Gamma = \{[x_i,]_{x_i}, x_i \mid i \in \mathbb{N}\}$.
- $[x_i$ and $]_{x_i}$ are called **parentheses**, x_i is called **variable**.
- A **ref-word over Σ** is a word $w \in (\Sigma \cup \Gamma)^*$.
- A ref-word is **valid** if, for every $i \in \mathbb{N}$,
 - ▶ only well-formed, non-nested pairs of parentheses $[x_i,]_{x_i}$,
 - ▶ no x_i inside of $[x_i \dots]_{x_i}$.
- $\Sigma^{[*]}$ is the set of **valid ref-words** (over Σ).

Reference-Words - Examples

- $[x [y b]_x c x [x b]_y z y b [y c z]_y z [z c c]_x]_z,$
- $[x a [y b [z b b a]_z c]_y b y b]_x x y.$

Reference-Words - Examples

- $[x [y b]_x]_x \text{ c x } [x b]_y \text{ z y b } [y c z]_y \text{ z } [z c c]_x]_z,$
- $[x a [y b [z b b a]_z c]_y \text{ b y b}]_x \text{ x y}.$

References for x with a value u : $[x u]_x$.

Reference-Words - Examples

- $[x [y b]_x c x [x b]_y z y b [y cz]_y z [z cc]_x]_z,$
- $[x a [y b [z bba]_z c]_y b y b]_x x y.$

References for x with a value u : $[x u]_x$.

An Occurrence of variable x **refers to** the reference for x , which precedes it.

Reference-Words - Examples

- $[_x [_y b]_x c]_x [_x b]_y zyb [_y cz]_y z [_z cc]_x]_z,$
- $[_x a [_y b [_z bba]_z c]_y byb]_x xy.$

References for x with a value u : $[_x u]_x$.

An Occurrence of variable x refers to the reference for x , which precedes it.

Undefined variables: x not preceded by a reference for x .

Reference-Words - Examples

- $[_x [_y b]_x c]_x [_x b]_y zyb [_y cz]_y z [_z cc]_x]_z,$
- $[_x a [_y b [_z bba]_z c]_y byb]_x xy.$

References for x with a value u : $[_x u]_x$.

An Occurrence of variable x refers to the reference for x , which precedes it.

Undefined variables: x not preceded by a reference for x .

Nested references: $[_x \dots [_y \dots]_y \dots]_x$.

Reference-Words - Examples

- $[_x [_y b]_x cx [_x b]_y zyb [_y cz]_y z [_z cc]_x]_z,$
- $[_x a [_y b [_z bba]_z c]_y byb]_x xy.$

References for x with a value u : $[_x u]_x$.

An Occurrence of variable x refers to the reference for x , which precedes it.

Undefined variables: x not preceded by a reference for x .

Nested references: $[_x \dots [_y \dots]_y \dots]_x$.

Overlapping references: $[_x \dots [_y \dots]_x \dots]_y$.

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$$za_{[z x [x y b_{[y c]_x} b x_{[x c]_y} b]_x y c]_z} x c z$$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$z a [z x [x y b [y c]_x b x [x c]_y b]_x y c]_z x c z$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$$a[z[xb[yc]_x]bx[xc]_y]_xyc]_zxcz$$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$$a[z[xb[yc]x]bx[xc]y]b]_xyc]_zxcz$$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$a[z[xb[yc]_x]_x]_x bbc[xc]_y b]_x yc]_z xcz$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$a[_z b[_y c b b c[_x c]_y b]_x y c]_z x c z$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zb_ycbbc_xc_yb_xyc_zxcz

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zb_ycbbc_xc_yb_xcbbcc_zxcz

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

$a[zbcbbc[xcb]_xcbbccc]_zxcz$

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zbcbbc_{[x cb]_x}cbbccc_zx_cz

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zbcbbc_{[x cb]_x}cbbccc_zcbcz

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_z[bcbbccbcbbccc]_zcbc_z

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zbcbbccbcbbccc_zcbc_z

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

a_zbcbbccbcbbccc_zcbc_zbcbbccbcbbccc

Reference-Words - Dereference Function

$$\mathcal{D} : \Sigma^{[*]} \rightarrow \Sigma^*$$

Example:

abcbbccbcbbccccbcbbccbcbbccc

Ref-Languages and Ref-Regular Languages

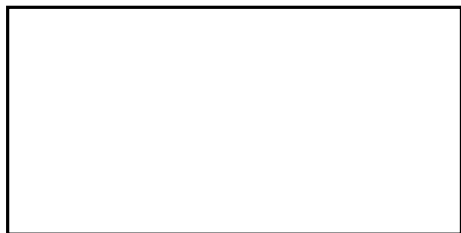
L is a **ref-language** (over Σ) if

- $L \subseteq \Sigma^{[*]}$ and
- $L \subseteq (\Sigma \cup \{[x_i,]_{x_i}, x_i \mid i \leq k\})^*$, for some $k \in \mathbb{N}$.

Ref-Languages and Ref-Regular Languages

L is a **ref-language** (over Σ) if

- $L \subseteq \Sigma^{[*]}$ and
- $L \subseteq (\Sigma \cup \{[x_i,]_{x_i}, x_i \mid i \leq k\})^*$, for some $k \in \mathbb{N}$.

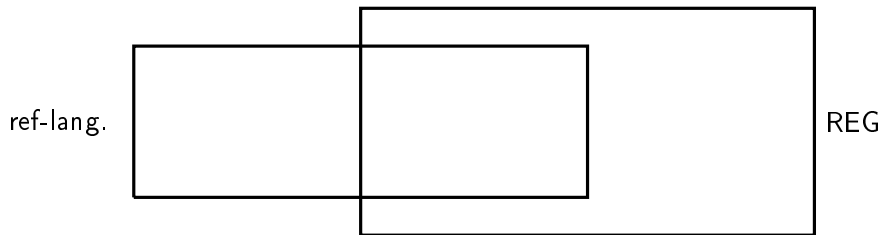


REG

Ref-Languages and Ref-Regular Languages

L is a **ref-language** (over Σ) if

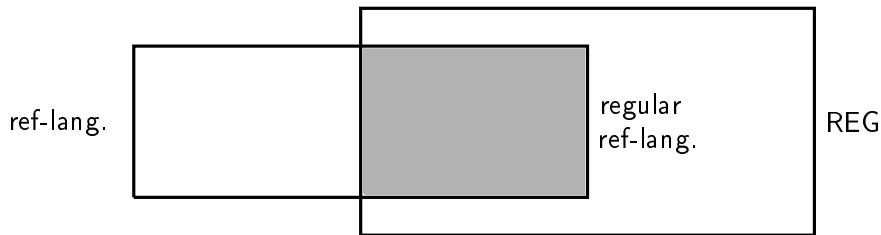
- $L \subseteq \Sigma^{[*]}$ and
- $L \subseteq (\Sigma \cup \{[x_i,]_{x_i}, x_i \mid i \leq k\})^*$, for some $k \in \mathbb{N}$.



Ref-Languages and Ref-Regular Languages

L is a **ref-language** (over Σ) if

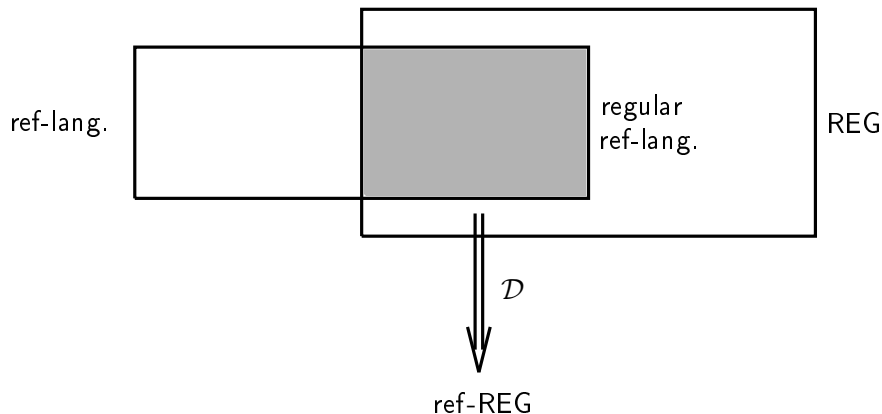
- $L \subseteq \Sigma^{[*]}$ and
- $L \subseteq (\Sigma \cup \{[x_i,]_{x_i}, x_i \mid i \leq k\})^*$, for some $k \in \mathbb{N}$.



Ref-Languages and Ref-Regular Languages

L is a **ref-language** (over Σ) if

- $L \subseteq \Sigma^{[*]}$ and
- $L \subseteq (\Sigma \cup \{[x_i,]_{x_i}, x_i \mid i \leq k\})^*$, for some $k \in \mathbb{N}$.



Ref-Languages and Ref-Regular Languages

- $\text{REG} \subset \text{ref-REG} \subset \text{CS}$,

Ref-Languages and Ref-Regular Languages

- $\text{REG} \subset \text{ref-REG} \subset \text{CS}$,
- $L_c = \{[x w]_x x \mid w \in \Sigma^*\}$ is a regular ref-language.
 $\mathcal{D}(L_c) = \{ww \mid w \in \Sigma^*\}$ (copy language).

Ref-Languages and Ref-Regular Languages

- $\text{REG} \subset \text{ref-REG} \subset \text{CS}$,
- $L_c = \{[x w]_x x \mid w \in \Sigma^*\}$ is a regular ref-language.
 $\mathcal{D}(L_c) = \{ww \mid w \in \Sigma^*\}$ (copy language).
- $\{a^n b^n \mid n \in \mathbb{N}\} \notin \text{ref-REG}$.

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1

M_2

M_3

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1

M_2

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1

M_2

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1

M_2

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: open M_1



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a

M_2

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: open M_2

M_1 a

M_2

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b

M_2 b

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: close M_1

M_1 a b c

M_2 b c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c c

M_3



a c a b c c **b a b c c b c c a b c c b c c a b c c**

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: close M_2

M_1 a b c

M_2 b c c

M_3



a c a b c c **b a b c c b c c a b c c b c c a b c c**

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: consult M_1

M_1 a b c

M_2 b c c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

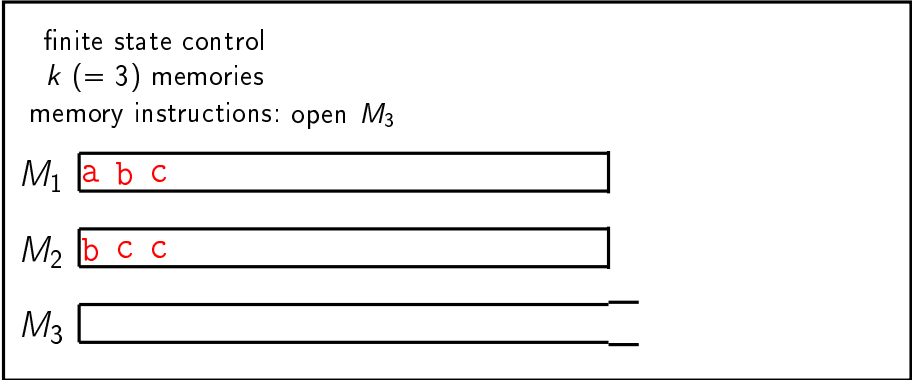
M_2 b c c

M_3



a c a b c c b a b c c **b c c a b c c b c c a b c c**

Memory Automata (MFA)



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: consult M_2

M_1 a b c

M_2 b c c

M_3



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c c

M_3 b c c



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: consult M_1

M_1 a b c

M_2 b c c

M_3 b c c



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 a b c

M_2 b c c

M_3 b c c a b c




a c a b c c b a b c c b c c a b c c b c c a b c c


Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: open M_1

M_1 

M_2 

M_3 



a c a b c c b a b c c b c c a b c **c b c c a b c c**

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 c

M_2 b c c

M_3 b c c a b c c



a c a b c c b a b c c b c c a b c c **b c c a b c c**

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: close M_3

M_1 c

M_2 b c c

M_3 b c c a b c c



a c a b c c b a b c c b c c a b c c **b c c a b c c**

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: consult M_3

M_1 c

M_2 b c c

M_3 b c c a b c c



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions:

M_1 c b c c a b c c

M_2 b c c

M_3 b c c a b c c



a c a b c c b a b c c b c c a b c c b c c a b c c

Memory Automata (MFA)

finite state control

$k (= 3)$ memories

memory instructions: close M_1

M_1 c b c c a b c c

M_2 b c c

M_3 b c c a b c c

a c a b c c b a b c c b c c a b c c b c c a b c c

Determinism of MFA

- **Pseudo deterministic**: For every state, symbol b and memory i , at most one move that reads b and at most one move that consults memory i .

Determinism of MFA

- **Pseudo deterministic:** For every state, symbol b and memory i , at most one move that reads b and at most one move that consults memory i .
- **Deterministic:** ϵ -free and for every state at most one possible move.

Determinism of MFA

- **Pseudo deterministic:** For every state, symbol b and memory i , at most one move that reads b and at most one move that consults memory i .
- **Deterministic:** ε -free and for every state at most one possible move.
- $\mathcal{L}(\text{MFA}) = \mathcal{L}(\text{pseudo-det- MFA})$ (extended subset construction).

Determinism of MFA

- **Pseudo deterministic**: For every state, symbol b and memory i , at most one move that reads b and at most one move that consults memory i .
- **Deterministic**: ϵ -free and for every state at most one possible move.
- $\mathcal{L}(\text{MFA}) = \mathcal{L}(\text{pseudo-det- MFA})$ (extended subset construction).
- $\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA})$ ($\{ww \mid w \in \{a, b\}^*\} \notin \mathcal{L}(\text{DMFA})$).

Nested MFA

An MFA is **nested**, if no two memories record factors that are overlapping, i. e.,

Nested MFA

An MFA is **nested**, if no two memories record factors that are overlapping, i. e.,

memory 2
a b a c b a c b a b c c b
memory 1

is not possible.

Nested MFA

An MFA is **nested**, if no two memories record factors that are overlapping, i. e.,

$$\begin{array}{c} \text{memory 2} \\ \underbrace{\hspace{10em}} \\ \text{a b a c b a c b a b c c b} \\ \underbrace{\hspace{5em}} \\ \text{memory 1} \end{array}$$

is not possible.

Theorem

*Every MFA(k) can be transformed into a equivalent MFA(k^2) that is pseudo-deterministic and **nested**.*

Equivalence of $\mathcal{L}(\text{MFA})$ and ref-REG

Theorem

ref-REG = $\mathcal{L}(\text{MFA})$.

Equivalence of $\mathcal{L}(\text{MFA})$ and ref-REG

Theorem

ref-REG = $\mathcal{L}(\text{MFA})$.

Define $\psi_{\mathcal{D}} : \{M \in \text{NFA} \mid L(M) \subseteq \Sigma^{[*]}\} \rightarrow \text{MFA}$ by

Equivalence of $\mathcal{L}(\text{MFA})$ and ref-REG

Theorem

ref-REG = $\mathcal{L}(\text{MFA})$.

Define $\psi_{\mathcal{D}} : \{M \in \text{NFA} \mid L(M) \subseteq \Sigma^{[*]}\} \rightarrow \text{MFA}$ by

NFA reads $a \in \Sigma$	\Rightarrow	MFA reads $a \in \Sigma$
NFA reads $[x_i$	\Rightarrow	MFA opens memory i
NFA reads $]x_i$	\Rightarrow	MFA closes memory i
NFA reads x_i	\Rightarrow	MFA consults memory i

Equivalence of $\mathcal{L}(\text{MFA})$ and ref-REG

Theorem

ref-REG = $\mathcal{L}(\text{MFA})$.

Define $\psi_{\mathcal{D}} : \{M \in \text{NFA} \mid L(M) \subseteq \Sigma^{[*]}\} \rightarrow \text{MFA}$ by

NFA reads $a \in \Sigma$	\Rightarrow	MFA reads $a \in \Sigma$
NFA reads $[x_i$	\Rightarrow	MFA opens memory i
NFA reads $]x_i$	\Rightarrow	MFA closes memory i
NFA reads x_i	\Rightarrow	MFA consults memory i

Lemma

Let $M \in \text{NFA}$ with $L(M) \subseteq \Sigma^{[*]}$. Then $\mathcal{D}(L(M)) = L(\psi_{\mathcal{D}}(M))$.

Equivalence of $\mathcal{L}(\text{MFA})$ and ref-REG

Theorem

ref-REG = $\mathcal{L}(\text{MFA})$.

Define $\psi_{\mathcal{D}} : \{M \in \text{NFA} \mid L(M) \subseteq \Sigma^{[*]}\} \rightarrow \text{MFA}$ by

NFA reads $a \in \Sigma$	\Rightarrow	MFA reads $a \in \Sigma$
NFA reads $[x_i$	\Rightarrow	MFA opens memory i
NFA reads $]x_i$	\Rightarrow	MFA closes memory i
NFA reads x_i	\Rightarrow	MFA consults memory i

Lemma

Let $M \in \text{NFA}$ with $L(M) \subseteq \Sigma^{[*]}$. Then $\mathcal{D}(L(M)) = L(\psi_{\mathcal{D}}(M))$.

Lemma

Let $M \in \text{MFA}$. Then $L(M) = \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M)))$.

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$$r := ((a | b)^*)(c^* | (a^*b))$$

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$r := (1(a|b)^*)_1(c^* | (2a^*b)_2)$

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$$r := (\color{red}{1} (a | b)^* \color{red}{1}) (\color{red}{1} c^* | (\color{blue}{2} a^* b \color{blue}{2})) (\color{blue}{2} | b^*) (\color{red}{1})^*$$

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$r := (\color{red}{1} (a | b)^* \color{red}{1}) (\color{red}{1} c^* | (\color{blue}{2} a^* b \color{blue}{2})) (\backslash \color{blue}{2} | b^*) (\backslash \color{red}{1})^*$

Some background information about REGEX:

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$$r := (\color{red}{1} (a | b)^* \color{red}{1}) (\color{red}{1} (c^* | (\color{blue}{2} a^* b \color{blue}{2})) (\color{blue}{2} | b^*) (\color{red}{1})^*$$

Some background information about REGEX:

- invented entirely on the level of software implementation,

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$r := (1(a|b)^*)_1(c^*|(2a^*b)_2)(\backslash 2|b^*)(\backslash 1)^*$

Some background information about REGEX:

- invented entirely on the level of software implementation,
- applied in practice: Traditional and Modern `grep`, `vi`, Modern `sed`, GNU Emacs, Perl, Python, Java, .Net,

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$r := (\color{red}{1} (a | b)^* \color{red}{1}) (\color{red}{1} c^* | (\color{blue}{2} a^* b \color{blue}{2})) (\backslash \color{blue}{2} | b^*) (\backslash \color{red}{1})^*$

Some background information about REGEX:

- invented entirely on the level of software implementation,
- applied in practice: Traditional and Modern `grep`, `vi`, Modern `sed`, GNU Emacs, Perl, Python, Java, .Net,
- *NP*-complete membership problem, undecidable inclusion problem,

Extended Regular Expressions with Backreferences (REGEX)

REGEX = regular expressions with references to subexpressions.

$$r := (\mathbf{1} (a | b)^* \mathbf{1}) (\mathbf{c}^* | (\mathbf{2} a^* b \mathbf{2})) (\backslash \mathbf{2} | b^*) (\backslash \mathbf{1})^*$$

Some background information about REGEX:

- invented entirely on the level of software implementation,
- applied in practice: Traditional and Modern `grep`, `vi`, Modern `sed`, GNU Emacs, Perl, Python, Java, .Net,
- *NP*-complete membership problem, undecidable inclusion problem,
- language theoretical investigation started 10 years ago (Câmpeanu, K. Salomaa, Yu).

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

Proof sketch:

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

Proof sketch:

$$r := (1(a|b)^*1)(c^*|(2a^*b)2)(\setminus 2|b^*)(\setminus 1)^*$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

Proof sketch:

$$r := (1 (a | b)^*)_1 (c^* | (2 a^* b)_2) (\backslash 2 | b^*) (\backslash 1)^*$$

$$r' := [x_1 (a | b)^*]_{x_1} (c^* | [x_2 a^* b]_{x_2}) (x_2 | b^*) (x_1)^*$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

Proof sketch:

$$r := (1 (a | b)^*)_1 (c^* | (2 a^* b)_2) (\setminus 2 | b^*) (\setminus 1)^*$$

$$r' := [x_1 (a | b)^*]_{x_1} (c^* | [x_2 a^* b]_{x_2}) (x_2 | b^*) (x_1)^*$$

$$L(r) = \mathcal{D}(L(r'))$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

$[x_2[x_1(a | b)^*]_{x_1}c^*x_1]_{x_2}ax_2x_1$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

$$[x_2[x_1(a | b)^*]_{x_1}c^*x_1]_{x_2}ax_2x_1 \Rightarrow (2(1(a | b)^*)_1c^*\backslash 1)_2a\backslash 2\backslash 1$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

$$[x_2 [x_1 (a \mid b)^*]_{x_1} c^* x_1]_{x_2} a x_2 x_1 \Rightarrow (([x_1 (a \mid b)^*]_{x_1}) c^* \setminus 1)_2 a \setminus 2 \setminus 1$$

$$(([x_1 a^*] \mid ([x_1 (a \mid b)^*])) ((ca]_{x_1} x_1) \mid]_{x_1}) x_1$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

$$[x_2 [x_1 (a \mid b)^*]_{x_1} c^* x_1]_{x_2} a x_2 x_1 \Rightarrow ((([x_1 (a \mid b)^*]_{x_1}) c^*)_{x_2}) a_{x_2} x_1$$

$$(([x_1 a^*]_{x_1}) \mid ([x_1 (a \mid b)^*]_{x_1})) ((c a]_{x_1} x_1) \mid]_{x_1}) x_1 \Rightarrow ???$$

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$?

A regular expression r with $L(r) \in \Sigma^{[*]}$ has the **REGEX property** if all $[x \dots]_x$ enclose a subexpression of r .

Let $L \in \text{ref-REG}$ and let r be a regular expression with $\mathcal{D}(L(r)) = L$. If r has the REGEX property, then $L \in \mathcal{L}(\text{REGEX})$.

$$[x_2 [x_1 (a | b)^*]_{x_1} c^* x_1]_{x_2} a x_2 x_1 \Rightarrow ({}_2({}_1(a | b)^*)_1 c^* \backslash 1)_2 a \backslash 2 \backslash 1$$

$$(([x_1 a^*] | ([x_1 (a | b)^*])) ((ca)_{x_1} x_1) |]_{x_1}) x_1 \Rightarrow ???$$

Question

Given a regular expression r with $L(r) \in \Sigma^{[*]}$. Is it possible to transform r into r' with the REGEX property and $\mathcal{D}(L(r)) = \mathcal{D}(L(r'))$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$.

Proof sketch:

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} \subseteq \mathcal{L}(\text{REGEX})$.

Proof sketch:

$L \in \text{ref-REG}$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$.

Proof sketch:

$L \in \text{ref-REG}$.

\exists nested MFA M with $L(M) = L$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$.

Proof sketch:

$L \in \text{ref-REG}$.

\exists nested MFA M with $L(M) = L$.

\exists NFA N with $L(N) = L' \in \Sigma^{[*]}$ and $\mathcal{D}(L') = L$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

ref-REG $\subseteq \mathcal{L}(\text{REGEX})$.

Proof sketch:

$L \in \text{ref-REG}$.

\exists nested MFA M with $L(M) = L$.

\exists NFA N with $L(N) = L' \in \Sigma^{[*]}$ and $\mathcal{D}(L') = L$.

Transform N into a regular expression r with $L(r) = L(N)$ that has the REGEX property.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

ref-regular languages are characterised by

- regular ref-languages,
 - ▶ finite automata accepting ref-languages,
 - ▶ regular expressions generating ref-languages,
 - ▶ ...

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

ref-regular languages are characterised by

- regular ref-languages,
 - ▶ finite automata accepting ref-languages,
 - ▶ regular expressions generating ref-languages,
 - ▶ ...
- MFA

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

ref-regular languages are characterised by

- regular ref-languages,
 - ▶ finite automata accepting ref-languages,
 - ▶ regular expressions generating ref-languages,
 - ▶ ...
- MFA
- REGEX (which can be considered a normal form of regular expressions generating ref-languages)

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

ref-regular languages are characterised by

- regular ref-languages,
 - ▶ finite automata accepting ref-languages,
 - ▶ regular expressions generating ref-languages,
 - ▶ ...
- MFA
- REGEX (which can be considered a normal form of regular expressions generating ref-languages)

We have characterisations of REGEX-languages independent of REGEX.

Equivalence of ref-REG and $\mathcal{L}(\text{REGEX})$

Theorem

$\text{ref-REG} = \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

ref-regular languages are characterised by

- regular ref-languages,
 - ▶ finite automata accepting ref-languages,
 - ▶ regular expressions generating ref-languages,
 - ▶ ...
- MFA
- REGEX (which can be considered a normal form of regular expressions generating ref-languages)

We have characterisations of REGEX-languages independent of REGEX.

We have separated the “regular”-part from the “reduplication”-part.

Deterministic MFA Languages

$\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA}) (= \mathcal{L}(\text{REGEX}) = \text{ref-REG})$.

Deterministic MFA Languages

$\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA}) (= \mathcal{L}(\text{REGEX}) = \text{ref-REG})$.

Theorem

The membership problem for DMFA-languages: $O(|w|)$.

Deterministic MFA Languages

$\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA}) (= \mathcal{L}(\text{REGEX}) = \text{ref-REG})$.

Theorem

The membership problem for DMFA-languages: $O(|w|)$.

Theorem

$\mathcal{L}(\text{DMFA})$ is closed under

- *complementation and*
- *intersection with regular languages,*

but it is not closed under

- *union or*
- *intersection.*

Deterministic MFA Languages

$\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA}) (= \mathcal{L}(\text{REGEX}) = \text{ref-REG})$.

Theorem

The membership problem for DMFA-languages: $O(|w|)$.

Theorem

The membership problem for ref-REG-languages: NP-complete.

Theorem

$\mathcal{L}(\text{DMFA})$ is closed under

- *complementation and*
- *intersection with regular languages,*

but it is not closed under

- *union or*
- *intersection.*

Deterministic MFA Languages

$\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA}) (= \mathcal{L}(\text{REGEX}) = \text{ref-REG})$.

Theorem

The membership problem for DMFA-languages: $O(|w|)$.

Theorem

The membership problem for ref-REG-languages: NP-complete.

Theorem

$\mathcal{L}(\text{DMFA})$ is closed under

- *complementation and*
- *intersection with regular languages,*

but it is not closed under

- *union or*
- *intersection.*

Theorem (Câmpeanu et al., Carle et al.)

ref-REG is closed under

- *union*
 - *intersection with regular languages,*
- but it is not closed under*
- *complementation or*
 - *intersection.*

Further Research Ideas

- Implementations of REGEX-engines based on MFA (or DMFA).

Further Research Ideas

- Implementations of REGEX-engines based on MFA (or DMFA).
- Descriptive complexity with respect to number of references in regular expression describing ref-languages, number of references in REGEX, number of memories of MFA.

Further Research Ideas

- Implementations of REGEX-engines based on MFA (or DMFA).
- Descriptive complexity with respect to number of references in regular expression describing ref-languages, number of references in REGEX, number of memories of MFA.
- Decision problems for ref-REG.

Further Research Ideas

- Implementations of REGEX-engines based on MFA (or DMFA).
- Descriptive complexity with respect to number of references in regular expression describing ref-languages, number of references in REGEX, number of memories of MFA.
- Decision problems for ref-REG.
- Investigate ref- \mathcal{L} for other language classes \mathcal{L} , e. g., ref-CF.

Thank you very much for your attention.