Pairing Systems

Lorenzo Laneve

Models of Computation

ETH Zurich

June 5th, 2020

Definition

A **pairing system** is a special **rewriting system**.

It consists of a tuple $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$.

Definition

A **pairing system** is a special **rewriting system**.

It consists of a tuple $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$.

- Σ is the alphabet of the input strings;

Definition

A **pairing system** is a special **rewriting system**.

It consists of a tuple $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$.

- Σ is the alphabet of the input strings;
- $\Gamma \supseteq \Sigma$ is the set of all the symbols used by the system;

Definition

A **pairing system** is a special **rewriting system**.

It consists of a tuple $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$.

- Σ is the alphabet of the input strings;
- $\Gamma \supseteq \Sigma$ is the set of all the symbols used by the system;
- $\mathcal{R}$ is an **ordered** set of rules of the form:

$$[a, b \to c]$$

  with $a, b, c \in \Gamma$.

Definition

A **pairing system** is a special **rewriting system**.

It consists of a tuple $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$.

- Σ is the alphabet of the input strings;
- $\Gamma \supseteq \Sigma$ is the set of all the symbols used by the system;
- $\mathcal{R}$ is an **ordered** set of rules of the form:

  $$[a, b \to c]$$

  with $a, b, c \in \Gamma$.
- $\mathcal{A} \subseteq \Gamma \cup \{\varepsilon\}$ is the set of **accepting symbols**, where ε is the empty string.

Configuration and Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The **configuration**, or **state** of the machine, at any point in time, is completely determined by a string $c \in \Gamma^*$.

Configuration and Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The **configuration**, or **state** of the machine, at any point in time, is completely determined by a string $c \in \Gamma^*$.

- The initial configuration is the input string $w \in \Sigma^*$.

Configuration and Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The **configuration**, or **state** of the machine, at any point in time, is completely determined by a string $c \in \Gamma^*$.

- The initial configuration is the input string $w \in \Sigma^*$.
- **String Rewriting System:** if a string contains occurrences of the substring ab, and we have a rule $[a, b \to c]$, then one of these occurrences will be replaced by c.

Configuration and Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The **configuration**, or **state** of the machine, at any point in time, is completely determined by a string $c \in \Gamma^*$.

- The initial configuration is the input string $w \in \Sigma^*$.
- **String Rewriting System:** if a string contains occurrences of the substring ab, and we have a rule $[a, b \rightarrow c]$, then one of these occurrences will be replaced by c.

But which occurrence exactly? And, if there is more than one rule applicable, which rule should be applied first?

Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

We define a **transition relation** between configurations $\vdash_{\mathcal{R}}$.

We say that the machine passes from a configuration $c \in \Gamma^*$ to another configuration $c' \in \Gamma^*$, if and only if:

$$c \vdash_{\mathcal{R}} c'$$

Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The following is true for $w_1, w_2 \in \Gamma^*$ and $X, Y, Z \in \Gamma$:

$$w_1 \cdot XY \cdot w_2 \vdash_{\mathcal{R}} w_1 \cdot Z \cdot w_2$$

if and only if:

Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The following is true for $w_1, w_2 \in \Gamma^*$ and $X, Y, Z \in \Gamma$:

$$w_1 \cdot XY \cdot w_2 \vdash_{\mathcal{R}} w_1 \cdot Z \cdot w_2$$

if and only if:

- $[X, Y \rightarrow Z]$ is the **first** applicable rule of $\mathcal{R}$ (remember, it is ordered!);

Transition

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

The following is true for $w_1, w_2 \in \Gamma^*$ and $X, Y, Z \in \Gamma$:

$$w_1 \cdot XY \cdot w_2 \vdash_{\mathcal{R}} w_1 \cdot Z \cdot w_2$$

if and only if:

- $[X, Y \rightarrow Z]$ is the **first** applicable rule of $\mathcal{R}$ (remember, it is ordered!);
- w_1 does not contain the substring XY, meaning that the occurrence we are replacing is the **leftmost** one.

Example of Computation

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

Consider the following pairing system:

- $\Sigma = \{a, b, c, d\}$
- $\Gamma = \{a, b, c, d, X, Z\}$
- $\mathcal{R} = ([c, d \to a], [a, b \to X], [X, X \to Z])$

and consider the input $\texttt{abcdb} \in \Sigma^*$:

Example of Computation

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

Consider the following pairing system:

- $\Sigma = \{a, b, c, d\}$
- $\Gamma = \{a, b, c, d, X, Z\}$
- $\mathcal{R} = ([c, d \to a], [a, b \to X], [X, X \to Z])$

and consider the input abcdb $\in \Sigma^*$:

$$\texttt{abcdb} \quad \vdash_{\mathcal{R}} \quad \texttt{abab}$$

Example of Computation

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

Consider the following pairing system:

- $\Sigma = \{a, b, c, d\}$
- $\Gamma = \{a, b, c, d, X, Z\}$
- $\mathcal{R} = ([c, d \to a], [a, b \to X], [X, X \to Z])$

and consider the input $\texttt{abcdb} \in \Sigma^*$:

$$\begin{aligned} \texttt{abcdb} &\vdash_{\mathcal{R}} \texttt{abab} \\ &\vdash_{\mathcal{R}} \texttt{Xab} \end{aligned}$$

Example of Computation

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

Consider the following pairing system:

- $\Sigma = \{a, b, c, d\}$
- $\Gamma = \{a, b, c, d, X, Z\}$
- $\mathcal{R} = ([c, d \to a], [a, b \to X], [X, X \to Z])$

and consider the input $\texttt{abcdb} \in \Sigma^*$:

$$\begin{aligned} \texttt{abcdb} &\vdash_{\mathcal{R}} \texttt{abab} \\ &\vdash_{\mathcal{R}} \texttt{Xab} \\ &\vdash_{\mathcal{R}} \texttt{XX} \end{aligned}$$

Example of Computation

$$\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$$

Consider the following pairing system:

- $\Sigma = \{a, b, c, d\}$
- $\Gamma = \{a, b, c, d, X, Z\}$
- $\mathcal{R} = ([c, d \to a], [a, b \to X], [X, X \to Z])$

and consider the input `abcdb` $\in \Sigma^*$:

$$\begin{aligned} \texttt{abcdb} &\vdash_{\mathcal{R}} \texttt{abab} \\ &\vdash_{\mathcal{R}} \texttt{Xab} \\ &\vdash_{\mathcal{R}} \texttt{XX} \\ &\vdash_{\mathcal{R}} \texttt{Z} \end{aligned}$$

Accepting a String

An input $w \in \Sigma^*$ is **reduced** to a single symbol $x \in \Gamma \cup \{\varepsilon\}$, i.e.:

$$w \vdash_{\mathcal{R}}^{*} x$$

where $\vdash_{\mathcal{R}}^{*}$ denotes the reflexive and transitive closure of $\vdash_{\mathcal{R}}$.

- We say that x is the **representative** of w.

Accepting a String

An input $w \in \Sigma^*$ is **reduced** to a single symbol $x \in \Gamma \cup \{\varepsilon\}$, i.e.:

$$w \vdash_{\mathcal{R}}^{*} x$$

where $\vdash_{\mathcal{R}}^{*}$ denotes the reflexive and transitive closure of $\vdash_{\mathcal{R}}$.

- We say that x is the **representative** of w.
- An input $w \in \Sigma^*$ is **accepted** by $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$ if and only if the representative of w is in $\mathcal{A}$, i.e.:

$$w \vdash_{\mathcal{R}}^{*} a \quad \text{for some } a \in \mathcal{A}$$

Observations on the model

- There may be strings without a representative (which are rejected)
  - e.g. no rules applicable and string still has more than one symbol.

Observations on the model

- There may be strings without a representative (which are rejected)
  - e.g. no rules applicable and string still has more than one symbol.
- But, if a representative exists, then it is **unique**.

Observations on the model

- There may be strings without a representative (which are rejected)
  - e.g. no rules applicable and string still has more than one symbol.
- But, if a representative exists, then it is **unique**.
- ε and symbols represent and are represented only by themselves.
  - No further rules are applicable to them.

Observations on the model

- There may be strings without a representative (which are rejected)
  - e.g. no rules applicable and string still has more than one symbol.
- But, if a representative exists, then it is **unique**.
- ε and symbols represent and are represented only by themselves.
  - No further rules are applicable to them.
- The computation on an input w stops after **at most** $|w| - 1$ steps.
  - As each rule decreases the length of the string by exactly 1.

Examples

We see three different languages:

- Regular language for emails
- Dyck language (well-formed parenthesys)
- A not-so-trivial regular language

Email Recognition

We want to recognize the following regular language:

$$\mathcal{L}_1 \equiv A^+ \texttt{@} A^+ . A^+$$

Let $\Sigma = \{A, \texttt{@}, .\}$ and $\Gamma = \{A, \texttt{@}, ., L, R, D, E\}$, with rules:

Email Recognition

We want to recognize the following regular language:

$$\mathcal{L}_1 \equiv A^+ @ A^+ . A^+$$

Let $\Sigma = \{A, @, .\}$ and $\Gamma = \{A, @, ., L, R, D, E\}$, with rules:

$$[A, @ \to L]$$

$$[A, L \to L] \qquad L \text{ represents } A^+ @$$

Email Recognition

We want to recognize the following regular language:

$$\mathcal{L}_1 \equiv A^+ @ A^+ . A^+$$

Let $\Sigma = \{A, @, .\}$ and $\Gamma = \{A, @, ., L, R, D, E\}$, with rules:

$$
\begin{array}{rl}
[A, @ \to L] & \\
[A, L \to L] & \quad L \text{ represents } A^+ @ \\
[L, A \to R] & \\
[R, A \to R] & \quad R \text{ represents } A^+ @ A^+
\end{array}
$$

Email Recognition

We want to recognize the following regular language:

$$\mathcal{L}_1 \equiv A^+ @ A^+ . A^+$$

Let $\Sigma = \{A, @, .\}$ and $\Gamma = \{A, @, ., L, R, D, E\}$, with rules:

$$\begin{array}{rl}
[A, @ \to L] & \\
[A, L \to L] & L \text{ represents } A^+@ \\
[L, A \to R] & \\
[R, A \to R] & R \text{ represents } A^+@A^+ \\
[R, . \to D] & D \text{ represents } A^+@A^+.
\end{array}$$

Email Recognition

We want to recognize the following regular language:

$$\mathcal{L}_1 \equiv A^+ @ A^+ . A^+$$

Let $\Sigma = \{A, @, .\}$ and $\Gamma = \{A, @, ., L, R, D, E\}$, with rules:

$[A, @ \to L]$	
$[A, L \to L]$	L represents $A^+ @$
$[L, A \to R]$	
$[R, A \to R]$	R represents $A^+ @ A^+$
$[R, . \to D]$	D represents $A^+ @ A^+ .$
$[D, A \to E]$	
$[E, A \to E]$	We set $\mathcal{A} = \{E\}$

Dyck Language

We want to recognize the context-free language represented by this grammar:

$$S \to SS \mid (S) \mid \varepsilon$$

Let $\Sigma = \{(,)\}$ and $\Gamma = \{(,), L, D\}$, with rules:

Dyck Language

We want to recognize the context-free language represented by this grammar:

$$S \rightarrow SS \mid (S) \mid \varepsilon$$

Let $\Sigma = \{(,)\}$ and $\Gamma = \{(,), L, D\}$, with rules:

$$[(,) \rightarrow D]$$
$$[D, D \rightarrow D]$$

Dyck Language

We want to recognize the context-free language represented by this grammar:

$$S \to SS \mid (S) \mid \varepsilon$$

Let $\Sigma = \{(,)\}$ and $\Gamma = \{(,), L, D\}$, with rules:

$$\begin{aligned} [(,) &\to D] \\ [D, D &\to D] \\ [(, D &\to L] \\ [L, D &\to L] \\ [L,) &\to D] \qquad \text{and set } \mathcal{A} = \{D, \varepsilon\} \end{aligned}$$

Dyck Language

We want to recognize the context-free language represented by this grammar:

$$S \to SS \mid (S) \mid \varepsilon$$

Let $\Sigma = \{(,)\}$ and $\Gamma = \{(,), L, D\}$, with rules:

$$\begin{aligned}
[(,) &\to D] \\
[D, D &\to D] \\
[(, D &\to L] \\
[L, D &\to L] \\
[L,) &\to D] \qquad \text{and set } \mathcal{A} = \{D, \varepsilon\}
\end{aligned}$$

- In this case, the last three rules are equivalent to $(D^*) \to D$.

Dyck Language

We want to recognize the context-free language represented by this grammar:

$$S \to SS \mid (S) \mid \varepsilon$$

Let $\Sigma = \{(,)\}$ and $\Gamma = \{(,), L, D\}$, with rules:

$$\begin{aligned} [(,) &\to D] \\ [D, D &\to D] \\ [(, D &\to L] \\ [L, D &\to L] \\ [L,) &\to D] \qquad \text{and set } \mathcal{A} = \{D, \varepsilon\} \end{aligned}$$

- In this case, the last three rules are equivalent to $(D^*) \to D$.
- Remember that D cannot represent ε!

A tedious language

We now look at the following regular language:

$$\mathcal{L}_3 \equiv (ccca)^* cacc (ccac)^*$$

A tedious language

We now look at the following regular language:

$$\mathcal{L}_3 \equiv (ccca)^* cacc(ccac)^*$$

Problem: we have three different strings $ccca$, $cacc$, $ccac$

A tedious language

We now look at the following regular language:

$$\mathcal{L}_3 \equiv (ccca)^* cacc (ccac)^*$$

Problem: we have three different strings *ccca*, *cacc*, *ccac*

- We would like to reduce to three different symbols...

A tedious language

We now look at the following regular language:

$$\mathcal{L}_3 \equiv (ccca)^* cacc (ccac)^*$$

Problem: we have three different strings $ccca$, $cacc$, $ccac$

- We would like to reduce to three different symbols...
- ...**but** reducing one of them inevitably "ruins" the others!

A tedious language

$$\mathcal{L}_3 \equiv (ccca)^* cacc(ccac)^*$$

Workaround:

A tedious language

$$\mathcal{L}_3 \equiv (ccca)^* cacc(ccac)^*$$

Workaround: we do the following rewriting

$$(ccca)^* \equiv cc(cacc)^* ca \cup \{\varepsilon\}$$

$$(ccac)^* \equiv c(cacc)^* cac \cup \{\varepsilon\}$$

And if we replace:

$$\mathcal{L}_3 \equiv (cc(cacc)^* ca \cup \{\varepsilon\}) \cdot cacc \cdot (c(cacc)^* cac \cup \{\varepsilon\})$$

A tedious language

$$\mathcal{L}_3 \equiv (cc(cacc)^* ca \cup \{\varepsilon\}) \cdot cacc \cdot (c(cacc)^* cac \cup \{\varepsilon\})$$

A tedious language

$$\mathcal{L}_3 \equiv (cc(cacc)^*ca \cup \{\varepsilon\}) \cdot cacc \cdot (c(cacc)^*cac \cup \{\varepsilon\})$$

If we now reduce *cacc*

- $[c, a \to E], [E, c \to F], [F, c \to G]$

A tedious language

$$\mathcal{L}_3 \equiv (cc(cacc)^* ca \cup \{\varepsilon\}) \cdot cacc \cdot (c(cacc)^* cac \cup \{\varepsilon\})$$

If we now reduce *cacc*

- $[c, a \to E], [E, c \to F], [F, c \to G]$

we obtain:

$$\begin{aligned}
\mathcal{L}_3 \equiv\ & (ccG^* E \cup \{\varepsilon\}) \cdot G \cdot (cG^* F \cup \{\varepsilon\}) \\
\equiv\ & \{ccG^* EGcG^* F\} \cup \\
& \{ccG^* EG\} \cup \\
& \{GcG^* F\} \cup \\
& \{G\}
\end{aligned}$$

Notes on Practicalities

- In most cases, modularity is easy and comes **natural**:
  - Reduce sub-languages to its representatives and then combine them.
- But sometimes it is not so easy...
  - Reducing a sub-language **may** erroneously touch other parts of the string.

Expressiveness of Pairing Systems

Which languages can we recognize? Which models can we simulate?

Expressiveness of Pairing Systems

Which languages can we recognize? Which models can we simulate?

Theorem

Pairing Systems are not Turing-complete.

This is immediate because Pairing Systems always terminate in a finite number of steps.

Expressiveness of Pairing Systems

Which languages can we recognize? Which models can we simulate?

Theorem

Pairing Systems are not Turing-complete.

This is immediate because Pairing Systems always terminate in a finite number of steps.

Corollary

There are recursively enumerable languages not recognizable by Pairing Systems.

Expressiveness of Pairing Systems

Theorem

For any regular language $\mathcal{L} \subseteq (\Sigma \setminus \{\mu\})^*$, *the language* $\mu\mathcal{L} \equiv \{\mu\ell \mid \ell \in \mathcal{L}\} \subseteq \Sigma^*$ *is recognizable by pairing systems.*

Expressiveness of Pairing Systems

Theorem

For any regular language $\mathcal{L} \subseteq (\Sigma \setminus \{\mu\})^*$, the language $\mu\mathcal{L} \equiv \{\mu\ell \,|\, \ell \in \mathcal{L}\} \subseteq \Sigma^*$ is recognizable by pairing systems.

Proof. Take a Finite State Automaton $\mathcal{F}$ for $\mathcal{L}$, with state space Q and accepting states set $F \subseteq Q$. We construct $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$:

Expressiveness of Pairing Systems

Theorem

For any regular language $\mathcal{L} \subseteq (\Sigma \setminus \{\mu\})^*$*, the language* $\mu\mathcal{L} \equiv \{\mu\ell \,|\, \ell \in \mathcal{L}\} \subseteq \Sigma^*$ *is recognizable by pairing systems.*

Proof. Take a Finite State Automaton $\mathcal{F}$ for $\mathcal{L}$, with state space Q and accepting states set $F \subseteq Q$. We construct $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$:

- $\Gamma = \Sigma \cup Q$, i.e. we use the states of $\mathcal{F}$ as symbols;

Expressiveness of Pairing Systems

Theorem

For any regular language $\mathcal{L} \subseteq (\Sigma \setminus \{\mu\})^*$*, the language* $\mu\mathcal{L} \equiv \{\mu\ell \,|\, \ell \in \mathcal{L}\} \subseteq \Sigma^*$ *is recognizable by pairing systems.*

Proof. Take a Finite State Automaton $\mathcal{F}$ for $\mathcal{L}$, with state space Q and accepting states set $F \subseteq Q$. We construct $\mathcal{S} = (\Sigma, \Gamma, \mathcal{R}, \mathcal{A})$:

- $\Gamma = \Sigma \cup Q$, i.e. we use the states of $\mathcal{F}$ as symbols;
- $\mathcal{A} \equiv F$, i.e. only accepting states of $\mathcal{F}$ are accepting symbols of $\mathcal{S}$.

Expressiveness of Pairing Systems

Assume without loss of generality that the FSA $\mathcal{F}$ is deterministic.

Expressiveness of Pairing Systems

Assume without loss of generality that the FSA $\mathcal{F}$ is deterministic.

If $\mathcal{F}$ switches from state $q_1 \in Q$ to state $q_2 \in Q$ upon reading symbol $a \in \Sigma$, we add the following rule to $\mathcal{R}$:

$$[q_1, a \to q_2]$$

replacing the initial state with μ.

Expressiveness of Pairing Systems

Assume without loss of generality that the FSA $\mathcal{F}$ is deterministic.

If $\mathcal{F}$ switches from state $q_1 \in Q$ to state $q_2 \in Q$ upon reading symbol $a \in \Sigma$, we add the following rule to $\mathcal{R}$:

$$[q_1, a \to q_2]$$

replacing the initial state with μ.

- The configuration will always be $q \cdot w$, where q is the current state of the FSA and w are the letters to read;
- The representative of μw is exactly the final state of $\mathcal{F}$ after reading w.

Expressiveness of Pairing Systems

Conjecture

There are context-free languages that cannot be recognized by Pairing Systems.

The following language does not seem to be recognizable:

$$\mathcal{L} \equiv \{ww^R \mid w \in \Sigma^*\}$$

where w^R denotes the reverse of w.

Notes on Expressiveness

- The model seems to be able to recognize all the context-free languages that have some **fixed structure** where we can start "eating symbols"
  - Emails have `@`, non-empty Dyck words have at least one ()

Notes on Expressiveness

- The model seems to be able to recognize all the context-free languages that have some **fixed structure** where we can start "eating symbols"
  - Emails have `@`, non-empty Dyck words have at least one ()
- **Conjecture**: all regular languages are recognizable
  - Maybe we can generalize the idea of the third example and apply an induction on regular expressions...

Pairing Systems

Thank you for your attention!