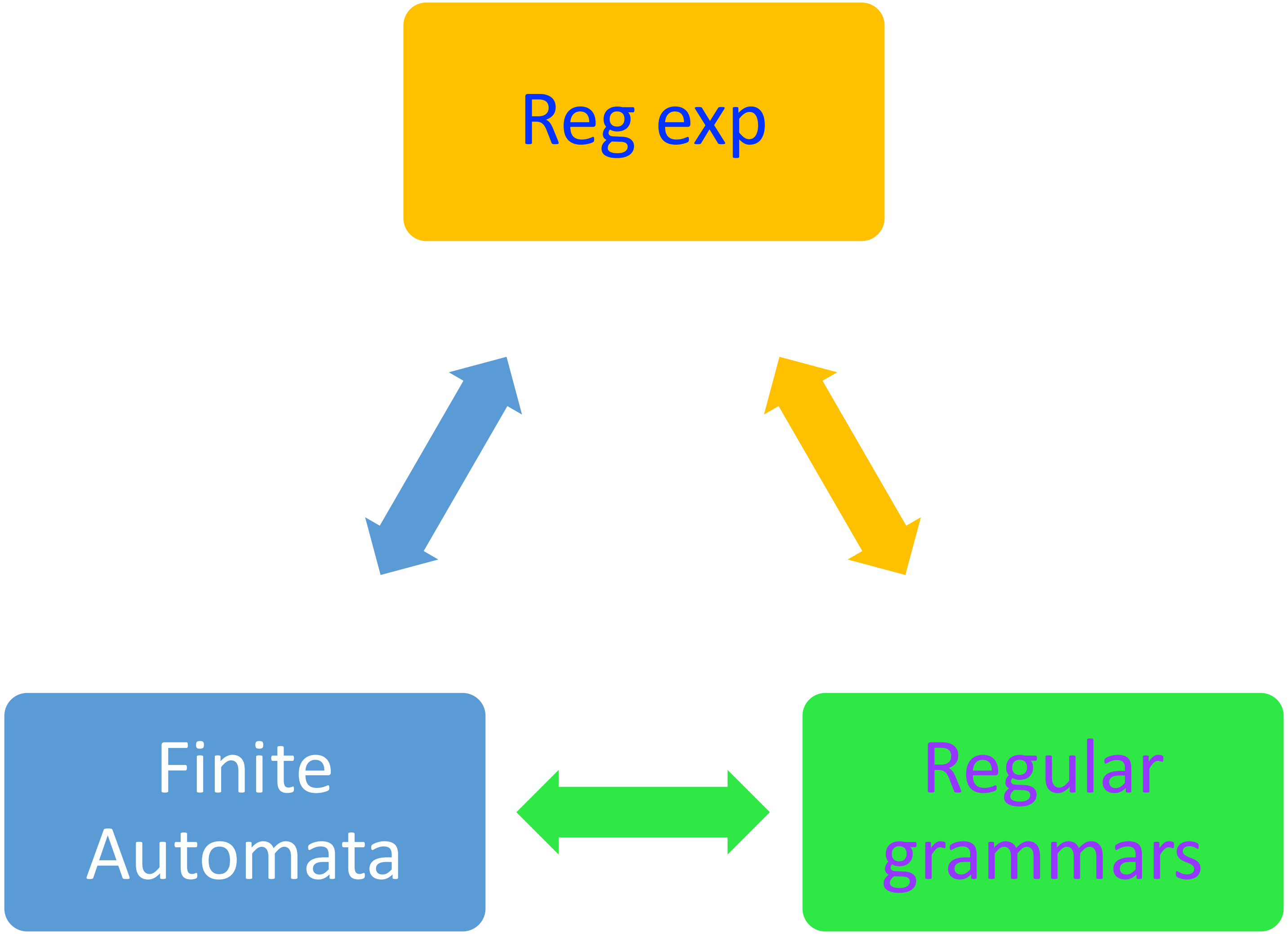
Course 5


Reg exp
Finite Automata
Regular grammars

Pumping Lemma

- Not all languages are regular
- How to decide if a language is regular or not?

- Idea: pump symbols

Example: $L = \{0^n 1^n \mid n >= 0\}$

Theorem: (Pumping lemma, Bar-Hillel)

Let L be a regular language. $\exists p \in N$, such that if $w \in L$ with $|w|>p$, then

$w = xyz$, where $0<|y|<=p$

and

$xy^iz \in L, \forall i \geq 0$

Proof

L regular => $\exists$ M = (Q,$\boldsymbol{\Sigma}$,$\boldsymbol{\delta}$, q_0, F) such that L= L(M)

Let |Q| = p

If w $\in$ L(M): (q_0,w) $\vdash^{*}$ (q_f , $\boldsymbol{\varepsilon}$) , $q_f \in$F

and

|w|>p

process at least p+1 symbols

p states

$\Rightarrow \exists$ q_1 that appear in at least 2 configurations

(q_0,xyz) $\vdash^{*}$ (q1,yz) $\vdash^{\pm}$ (q1,z) $\vdash^{*}$ (q_f , $\boldsymbol{\varepsilon}$) , $q_f \in$F => 0<=|y|<=p

Proof (cont)

$(q_0,xy^iz) \overset{*}{\vdash} (q_1,y^iz)$

$\overset{*}{\vdash} (q_1,y^{i-1}z)$

$\overset{*}{\vdash} \ldots$

$\overset{*}{\vdash} (q_1,yz)$

$\overset{*}{\vdash} (q_1, z)$

$\overset{*}{\vdash} (q_f , \boldsymbol{\varepsilon}) , q_f \in F$

So, if $w=xyz \in L$ then $xy^iz \in L$, for all $i>0$

If $i=0$: $(q_0,xz) \overset{*}{\vdash} (q_1, z) \overset{*}{\vdash} (q_f , \boldsymbol{\varepsilon}) , q_f \in F$

Example: $L = \{0^n 1^n \mid n >= 0\}$

Suppose L is regular => $w = xyz = 0^n 1^n$

Consider all possible decomposition =>

Case 1. $y = 0^k$

$$xyz = 0^{n-k} 0^k 1^n;\ xy^i z = 0^{n-k} 0^{ik} 1^n \notin L$$

Case 2. $y = 1^k$

$$xyz = 0^n 1^k 1^{n-k};\ xy^i z = 0^n 1^{ik} 1^{n-k} \notin L$$

Case 3. $y = 0^k 1^l$

$$xyz = 0^{n-k} 0^k 1^l 1^{n-l};\ xy^i z = 0^{n-k} (0^k 1^l)^i 1^{n-l} \notin L$$

Case 4. $y = 0^k 1^K$

$$xyz = 0^{n-k} 0^k 1^k 1^{n-k};\ xy^i z = 0^{n-k} 0^k 1^k 0^k 1^k \ldots 1^{n-l} \notin L$$

=> L is not regular

Context free grammar (cfg)

- Produtions of the form: A $\rightarrow\alpha$, A$\in$N, $\alpha\in(N\cup\boldsymbol{\Sigma})^*$
- More powerful
- Can model programming language:

  G = (N, $\boldsymbol{\Sigma}$,P,S) s.t. L(G) = programming language

Syntax tree

Definition: A syntax tree corresponding to a cfg $G = (N, \Sigma, P, S)$ is a tree obtained in the following way:

1. Root is the starting symbol S
2. Nodes $\in N \cup \Sigma$:
    1. Internal nodes $\in N$
    2. Leaves $\in \Sigma$
3. For a node A the descendants in order from left to right are $X_1, X_2, \ldots, X_n$ only if $A \rightarrow X_1X_2\ldots X_n \in P$

Remarks:

a) Parse tree = syntax tree – result of parsing (syntatic analysis)

b) Derivation tree – condition 2.2 not satisfied

c) Abstract syntax tree (AST) ≠ syntax tree (semantic analysis)

Syntax tree (cont)

Property: In a cfg $G = (N, \Sigma, P, S)$, $w \in L(G)$ <u>if and only if</u> there exists a syntax tree with frontier w.

Proof: HW

Definition: A cfg G = (N, Σ,P,S) is ambigous if for a w ∊ L(G) there exists 2 distinct syntax tree with frontier w.

Example:

Parsing (syntax analysis) modeled with cfg:

cfg $G = (N, \boldsymbol{\Sigma}, P, S)$:

- N – nonterminal: syntactical constructions: declaration, statement, expression, a.s.o.
- $\boldsymbol{\Sigma}$ – terminals; elements of the language: identifiers, constants, reserved words, operators, separators
- P – syntactical rules – expressed in BNF – simple transformation
- S – syntactical construct corresponding to program

THEN

Program syntactical correct $\Leftrightarrow w \in L(G)$

Back to compiler construction