TDDD55 Compilers and interpreters

TDDE66 Compiler Construction

LINKÖPINGS UNIVERSITET

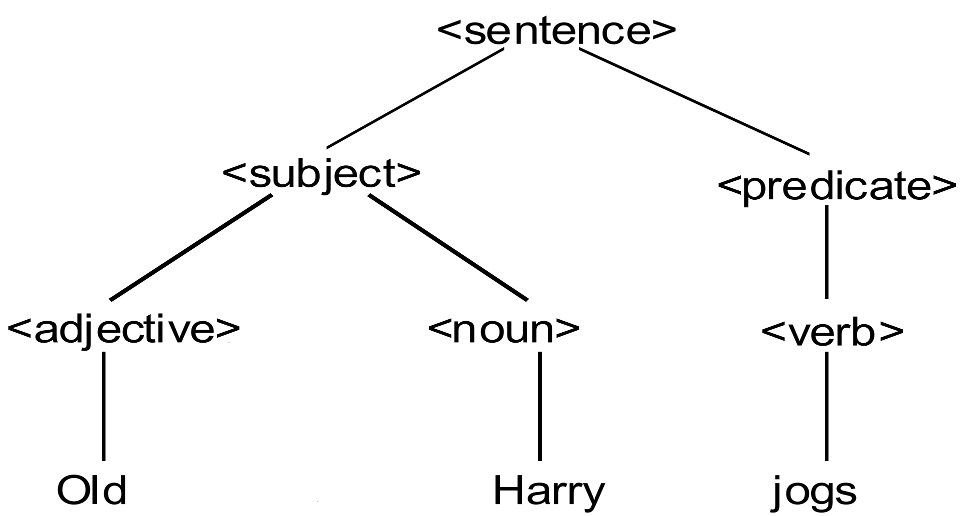
Formal Languages Part 2
Context Free Grammars

IDA, Linköpings universitet, 2024

Context-Free Grammars

- Example: an English sentence:

Sentence:	**Old**	**Harry**	**jogs**
Constituents:	subject		predicate
Word Class:	adjective	noun	verb



- A grammar is used to describe the syntax.

  BNF (Backus-Naur form) 1960 (meta-language to describe languages):

  - <sentence> $\rightarrow$ <subject><predicate>
  - <subject> $\rightarrow$ <adjective> <noun>
  - <predicate> $\rightarrow$ <verb>
  - <adjective> $\rightarrow$ old | big | strong | ...
  - <noun> $\rightarrow$ Harry | brother | ...
  - <verb> $\rightarrow$ jogs | snores | sleeps | ...

Grammars, cont.

LINKÖPINGS UNIVERSITET

- <sentence> is a *start symbol.*
- *Symbols to the left of "→"* are called *nonterminals*.
- Symbols not surrounded by "< >" are *terminals.*
- *Each line is a production.*

Symbol	Meaning
< ... >	syntactic classes
→	"consists of", "is" (also "::=")
|	"or"

A Grammar can be used to Produce or Derive Sentences

- Example: <sentence> $\Rightarrow^*$ Old Harry jogs
  - where <sentence> is the start symbol and $\Rightarrow^*$ means derivation in zero or more steps.

Example Derivation:

<sentence> $\Rightarrow$ <subject> <predicate>
$\Rightarrow$ <adjective> <noun> <predicate>
$\Rightarrow$ Old <noun> <predicate>
$\Rightarrow$ Old Harry <predicate>
$\Rightarrow$ Old Harry <verb>
$\Rightarrow$ Old Harry jogs

Definition: CFG (Context-free grammar)

- A CFG (Context-free grammar) is a quadruple (4 parts):

  $G = < N, \sum, P, S >$

  where
  N : Nonterminals.
  $\sum$: terminal Symbols.
  P : rules, Productions of the form
  $A \rightarrow a$ where $A \in N$ and
  $a \in (N \cup \sum)^*$
  S : the Start symbol,
  a nonterminal, $S \in N$.

- (Sometimes $V = N \cup \sum$ is used, called the *vocabulary*.)

- Example:
  - 1. <number> → <no>
  - 2. <no> → <no> <digit>
  - 3. | <digit>
  - 4. <digit> → 0|1|2|3|4|5|6|7|8|9

- N = { <number>, <no>, <digit> }
- $\sum$ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- S = <number>

Notational Conventions

$\alpha,\beta,\gamma \in V^*$	string of terminals and nonterminals
$A, B, C \in N$	nonterminals
$a, b, c \in \Sigma$	terminal symbols
$u, v, w, x, y, z \in \Sigma^*$	string of terminals

Derivations

Example Grammar:

1. <number> $\rightarrow$ <no>
2. <no> $\rightarrow$ <no> <digit>
3. | <digit>
4. digit> $\rightarrow$ 0|1|2|3|4|5|6|7|8|9

- **Derivation**
  - α $\Rightarrow \beta$ (pronounced "α derives β")
  - Formally: γ A $\theta \Rightarrow \gamma\, \delta\, \theta$ if we have A $\rightarrow \delta$
  - Example: <number> $\Rightarrow_{rm}$ <no> $\Rightarrow_{rm}$ <no> <digit> $\Rightarrow_{rm}$ <no> 2 $\Rightarrow_{rm}$ <digit> 2 $\Rightarrow_{rm}$ 12
  - <number> $\Rightarrow$ <no> direct derivation.
  - <number> $\Rightarrow^*$ 12 several derivations (zero or more).
  - <number> $\Rightarrow^+$ 12 several derivations (one or more).

- Given G = < N, ∑, P, S > the language generated by G can be defined as L(G):

$$L(G) = \{ w \mid S \Rightarrow^+ w \text{ and } w \in \Sigma^* \}$$

Sentential form, Sentence

LINKÖPINGS UNIVERSITET

Sentential form

- A string α is a *sentential form* in G if
- $S \Rightarrow^* \alpha$ and $\alpha \in V^*$ (string of terminals and/or nonterminals)
- Example: <no> <digit> is a sentential form in G(<number>). <no>8 is another sentential form

Sentence

- w is a *sentence* in G if $S \Rightarrow^+ w$ and $w \in \Sigma^*$.
- Example: 12 is a sentence in G(<number>).

Left and Right Derivations

- **Left derivation**
  - $\Rightarrow_{lm}$ means that we replace the *leftmost* nonterminal by some appropriate right side.
- **Left sentential form**
  - A sentential form which is part of a leftmost derivation.

- **Right derivation (canonical derivation)**
  - $\Rightarrow_{rm}$ means that we replace the *rightmost* nonterminal by some appropriate right side.
- **Right sentential form**
  - A sentential form which is part of a rightmost derivation.

Rightmost Derivation, Handle

- **Reverse rightmost derivation**
  - 12 <=$_{rm}$ <digit> 2 <=$_{rm}$ <no> 2 <=$_{rm}$ <no> <digit> <=$_{rm}$ <no> <=$_{rm}$ <number>
- **Handles**
  Consist of two parts:
  - 1. A production $A \rightarrow \beta$
  - 2. A position
  - If $S \Rightarrow^{*}_{rm} \alpha A w \Rightarrow_{rm} \alpha \beta w$, the production $A \rightarrow \beta$ together with the position after α is a **handle** of $\alpha \beta w$.

1. <number> → <no>
2. <no> → <no> <digit>
3. | <digit>
4. digit> → 0|1|2|3|4|5|6|7|8|9

- Example: The handle of <u><no> 2</u> is the production <digit> → 2 and the position after <no> because:
  - <number> $\Rightarrow_{rm}$ <no> $\Rightarrow_{rm}$ <no> <digit> $\Rightarrow_{rm}$ <no> 2 $\Rightarrow_{rm}$ <digit> 2 $\Rightarrow_{rm}$ 12

- Informally: a handle is what we *reduce* to what and where to get the previous sentential form in a rightmost derivation.

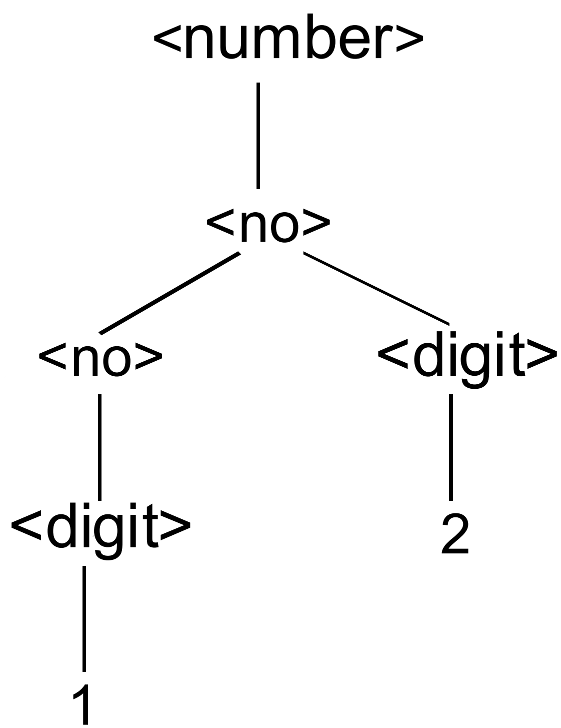
Reduction

- Reduction of a grammar rule

  In reverse right derivation, find a **right side** in some rule according to the grammar in the given right sentential form and **replace** it with the corresponding **left side**, i.e., nonterminal

Parse trees (derivation trees)

- A parse tree can correspond to several different derivations.



Parse tree for 12

Example Grammar:

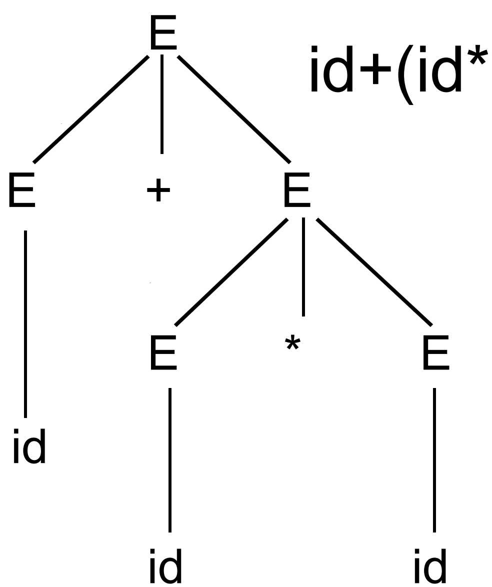
1. <number> $\rightarrow$ <no>
2. <no> $\rightarrow$ <no> <digit>
3. | <digit>
4. digit> $\rightarrow$ 0|1|2|3|4|5|6|7|8|9

Ambiguous Grammars

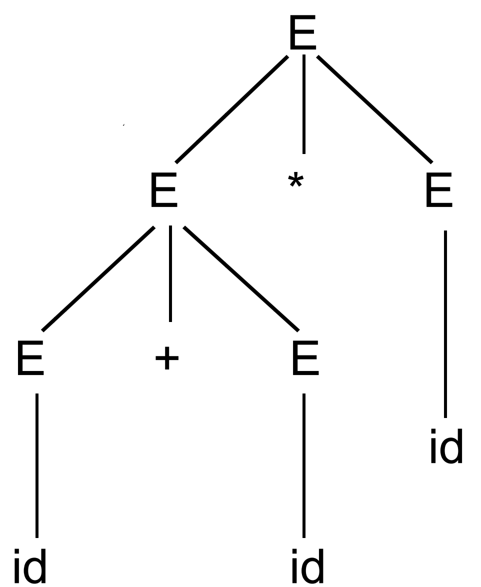
- A grammar G is *ambiguous* if a sentence in G has several different parse trees.

  e.g. $E \rightarrow E + E \mid E * E \mid E \uparrow E \mid id$

- id+id*id has two different parse trees.



id+(id*id)



(id+id)*id

Rewriting to Unambiguous Grammar

- Rewrite the grammar to make it unambiguous:
  - +, * are to have the right priority and
  - +, * are to be left associative while
  - ↑ is to be right associative.

- Example: a+b+c+d is interpreted as (a+b)+c+d, using the rewritten grammar:

E	→	E + T	(left associative)
	|	T	
T	→	T * F	(left associative)
	|	F	
F	→	P ↑ F	(right associative)
	|	P	
P	→	id	

Small Parse Tree Exercise

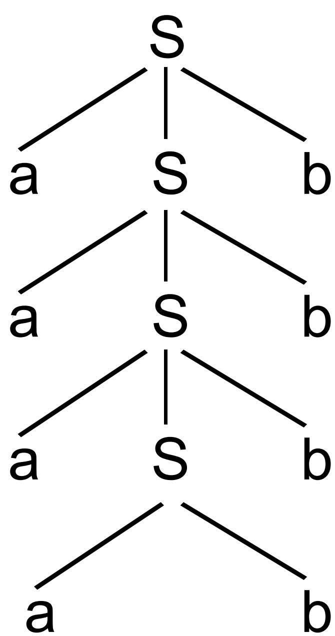
See 00-LectureExercises

Example Palindrome Grammars

- Palindrome: a string that is symmetrical around its center
- Example: The following grammar generates
  $\{ a^n b^n \mid n >= 1 \}$.

$$
\begin{aligned}
S \;&\rightarrow\; a\,S\,b \\
&\mid\; a\,b
\end{aligned}
$$

Example parse tree:



Another example:
Grammar describing binary palindromes of *odd* lengths >= 1:

$$
\begin{aligned}
S \rightarrow\; & 0\,S\,0 \\
\mid\; & 1\,S\,1 \\
\mid\; & 0 \\
\mid\; & 1
\end{aligned}
$$

Example derived strings: 0, 1, 000, 010, 111

Binary Palindrome Exercise

See 00-LectureExercises

Example
Excerpt from a Pascal Grammar

```
goal>          →  <progdecl> .
<progdecl>     →  <prog_hedr> ; <block>
<prog_hedr>    →  program <idname>
                   ( <idname_list> )
                |  program <idname>
<block>        →  <decls> begin
                    <stat_list>
                   end
<decls>        →  <labels> <consts>
                   <types> <vars> <procs>
<labels>       →  label <label_decl> ;
                |  ε
<label_decl>   →  <label_decl> , <labelid>
                |  <labelid>
<labelid>      →  <int>
                    | <id>
<consts>       →  const <const_decls>
                    | ε
```

```
<const_decls>   →  <const_decls> <const_decl_c>
                 |  <const_decl_c>
<const_decl_c>  →  <const_decl> ;
<const_decl>    →  <idname> = <const>
<types>         →  type <type_decls>
                 |  ε
<type_decls>    →  <type_decls> <type_decl_c>
                 |  <type_decl_c>
<type_decl_c>   →  <type_decl> ;
<type_decl>     →  <idname> = <type>
<vars>          →  var <var_decls>
                 |  ε
<var_decls>     →  <var_decls> <var_decl_c>
                 |  <var_decl_c>
<var_decl_c>    →  <var_decl> ;
<var_decl>      →  <id_list> : <type>
<procs>         →  <proc_decls>
                 |  ε
<proc_decls>    →  <proc_decls> <proc>
                 |  <proc>
```

Example
Excerpt from a Pascal Grammar, Cont.

```
<proc>       → procedure <phead_c> forward ;
               | procedure <phead_c> <block> ;
               | function <fhead_c> forward ;
               | function <fhead_c> <block> ;

<fhead_c>    → <fhead> ;

<fhead>      → <idname> <params> : <type_id>

<phead_c>    → <phead> ;

<phead>      → <idname> <params>
               | ϵ
<params>     → ( <param_list> )
               | ϵ
<param>      → var <par_decl>
               | <par_decl>
               | ϵ
<par_decl>   → <id_list> : <type_id>
<param_list> → <param_list> ; <param>
               | <param>
<id_list>    →  <id_list> , <id>
               | <id>
...
```