Syntax Analysis

– Context-Free Grammars –

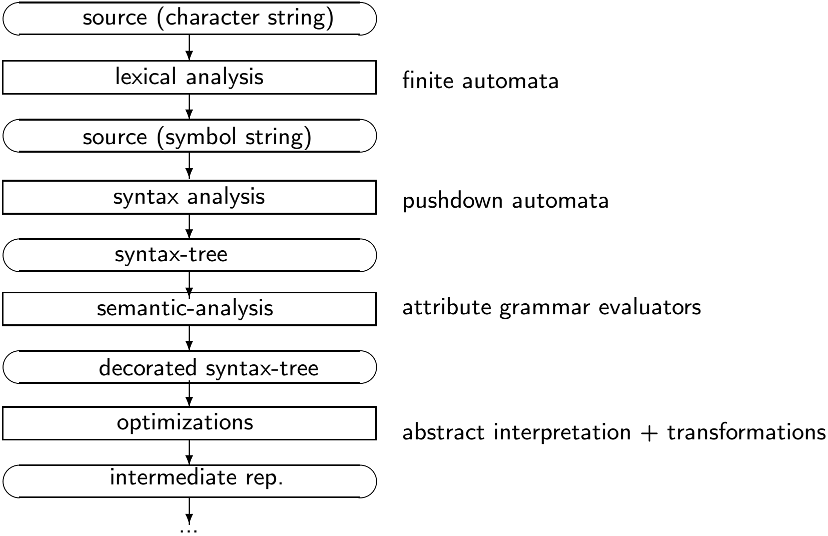
– Wilhelm/Seidl/Hack: Compiler Design, Syntactic and Semantic Analysis–

Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-saarland.de
and
Mooly Sagiv
Tel Aviv University
sagiv@math.tau.ac.il

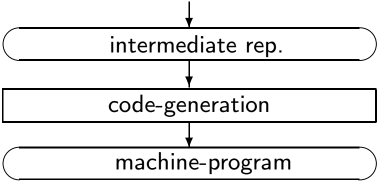
Subjects

- Introduction
  - The task of syntax analysis
  - Automatic generation
  - Error handling
- Context free grammars, derivations, and parse trees
- Pushdown automata
- Top-down syntax analysis
- Bottom-up syntax analysis - only a sketch

"Standard" Structure



"Standard" Structure cont'd



tree automata + dynamic programming + $\cdots$

Syntax Analysis (Parsing)

- Functionality

  Input Sequence of symbols (tokens)

  Output Parse tree

- Report syntax errors, e,g., unbalanced parentheses
- Create "‘pretty-printed" version of the program (sometimes)
- In many cases the tree need not be generated (one-pass compilers)

Note: Input is considered as a word over a new (finite) alphabet, i.e. the set of all symbol classes.

Handling Syntax Errors

- Report and locate the error (symptom)
- Diagnose the error
- Correct the error
- Recover from the error in order to discover more errors (without reporting too many follow up errors)

Example

$$a := a * (b + c * d;$$

The Valid Prefix Property

- For every word u that the parser identifies as a legal prefix, there exists a word w such that uw is a valid program — u has a continuation w
- Property of a parsing method
- All the parsing methods treated, i.e. LL-parsing and LR-parsing, have the valid prefix property.

Error Diagnosis Data

- Line number (may be far from the actual error)
- The current symbol
- The symbols expected in the current parser state
- Parser configuration

Error Recovery

- Becomes less important in interactive environments
- Example heuristics:
  - Search for a "significant" symbol and ignore the string up to this symbol (*panic mode*)
  - Try to "replace" symbols for common errors
  - Refrain from reporting more than 3 subsequent errors
- Globally optimal solutions — For every illegal input w, find a legal input w' with a "minimal distance" from w

Example Context Free Grammar (Statement Part)

Stat	$\rightarrow$	If_Stat |
		While_Stat |
		Repeat_Stat |
		Proc_Call |
		Assignment
If_Stat	$\rightarrow$	**if** Cond **then** Stat_Seq **else** Stat_Seq **fi** |
		if Cond **then** Stat_Seq **fi**
While_Stat	$\rightarrow$	**while** Cond **do** Stat_Seq **od**
Repeat_Stat	$\rightarrow$	**repeat** Stat_Seq **until** Cond
Proc_Call	$\rightarrow$	Name **(** Expr_Seq **)**
Assignment	$\rightarrow$	Name := Expr
Stat_Seq	$\rightarrow$	Stat |
		Stat_Seq; Stat
Expr_Seq	$\rightarrow$	Expr |
		Expr_Seq, Expr

Context-Free-Grammar Definition

A context-free-grammar is a quadruple $G = (V_N, V_T, P, S)$ where:

- V_N — finite set of non-terminals
- V_T — finite set of terminals
- $P \subseteq V_N \times (V_N \cup V_T)^*$ — finite set of production rules
- $S \in V_n$ — the start non-terminal

- A production $(A, \alpha) \in P$ is written as $A \rightarrow \alpha$
- read as " A may be derived to α" or
- as "α may be reduced to A"

Examples

$$G_0 = (\{E, T, F\}, \{+, *, (,), \mathbf{id}\}, P, E)$$

$$P = \left\{ \begin{array}{lcl} E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid \mathbf{id} \end{array} \right\}$$

$$G_1 = (\{E\}, \{+, *, (,), \mathbf{id}\}, \{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}\}, E)$$

G_0 and G_1 generate the same language.
What is the difference between the two grammars?

Derivations

Given a context-free-grammar $G = (V_N, V_T, P, S)$

- A derivation step $\varphi \implies \psi$

  if there exist $\varphi_1, \varphi_2 \in (V_N \cup V_T)^*$, $A \in V_N$
  - $\varphi \equiv \varphi_1 \, A \, \varphi_2$
  - $A \rightarrow \alpha \in P$
  - $\psi \equiv \varphi_1 \, \alpha \, \varphi_2$
- $\varphi \overset{*}{\implies} \psi$ reflexive transitive closure
- The language defined by G

$$L(G) = \{w \in V_T^* \mid S \overset{*}{\implies} w\}$$

Reduced and Extended Context Free Grammars

A non-terminal A is

reachable: There exist φ_1, φ_2 such that $S \stackrel{*}{\Longrightarrow} \varphi_1 A \varphi_2$

productive: There exists $w \in V_T^*$, $A \stackrel{*}{\Longrightarrow} w$

Removal of unreachable and unproductive non-terminals and the productions they occur in doesn't change the defined language.

A grammar is reduced if it has neither unreachable nor unproductive non-terminals.

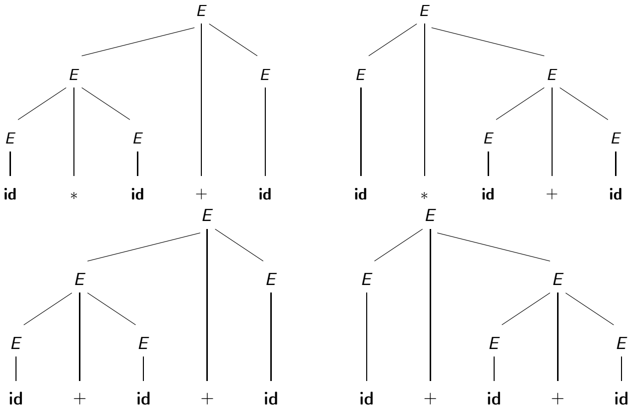
A grammar is extended if a new startsymbol S' and a new production $S' \rightarrow S$ are added to the grammar.

From now on, we only consider reduced and extended grammars.

Syntax-Tree (Parse-Tree)

- An ordered tree.
- Root is labeled with S.
- Internal nodes are labeled by non-terminals.
- Leaves are labeled by terminals or by ε.
- For internal nodes n: Is n labeled by N and are its children $n.1, n.2, \ldots, n.n_p$ labeled by $N_1, N_2, \ldots, N_{n_p}$, then $N \rightarrow N_1 N_2 \ldots N_{n_p} \in P$.

Examples



Leftmost (Rightmost) Derivations

Given a context-free-grammar $G = (V_N, V_T, P, S)$

- $\varphi \underset{lm}{\Longrightarrow} \psi$ if there exist $\varphi_1 \in V_T^*$, $\varphi_2 \in (V_N \cup V_T)^*$, and $A \in V_N$
  - $\varphi \equiv \varphi_1 \, A \, \varphi_2$
  - $A \rightarrow \alpha \in P$
  - $\psi \equiv \varphi_1 \, \alpha \, \varphi_2$

  replace leftmost non-terminal
- $\varphi \underset{rm}{\Longrightarrow} \psi$ if there exist $\varphi_2 \in V_T^*$, $\varphi_1 \in (V_N \cup V_T)^*$, and $A \in V_N$
  - $\varphi \equiv \varphi_1 \, A \, \varphi_2$
  - $A \rightarrow \alpha \in P$
  - $\psi \equiv \varphi_1 \, \alpha \, \varphi_2$

  replace rightmost non-terminal
- $\varphi \overset{*}{\underset{lm}{\Longrightarrow}} \psi$, $\varphi \overset{*}{\underset{rm}{\Longrightarrow}} \psi$ are defined as usual

Ambiguous Grammar

A grammar that has (equivalently)

- two leftmost derivations for the same string,
- two rightmost derivations for the same string,
- two syntax trees for the same string.