Syntactic Analysis

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

25. Oktober 2011
Subjects

- Introduction
  - The task of syntax analysis
  - Automatic generation
  - Error handling
- Context free grammars, derivations, and parse trees
- Grammar Flow Analysis
- Pushdown automata
- Top-down syntax analysis
- Bottom-up syntax analysis
- Bison — A parser generator
“Standard” Structure

source (character string)

lexical analysis Vol.2, Ch.2

source (symbol string)

syntax analysis Vol.2, Ch.3

syntax-tree

semantic-analysis Vol.2, Ch.4

decorated syntax-tree

optimizations Vol.3

intermediate rep.

finite automata

pushdown automata

attribute grammar evaluators

abstract interpretation + transformations
"Standard" Structure cont'd

intermediate rep. → code-generation Vol.4 → machine-program → tree automata + dynamic programming + ...
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

\[ a := a \times (b + c) \times d; \]
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 (Section)

Stat → If_Stat | While_Stat | Repeat_Stat | Proc_Call | Assignment

If_Stat → if Cond then Stat_Seq else Stat_Seq fi | if Cond then Stat_Seq fi

While_Stat → while Cond do Stat_Seq od

Repeat_Stat → repeat Stat_Seq until Cond

Proc_Call → Name ( Expr_Seq )

Assignment → Name := Expr

Stat_Seq → Stat | Stat_Seq; Stat

Expr_Seq → 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 nonterminals
- $V_T$ — 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 nonterminal
Examples

\[ G_0 = (\{E, T, F\}, \{+, *, (,), id\}, \]
\[
\{ E \rightarrow E + T \mid T \\
T \rightarrow T * F \mid F \ E \}
\]
\[
F \rightarrow (E) \mid id\}, \\
G_1 = (\{E\}, \{+, *, (,), id\}, \{E \rightarrow E + E \mid E * E \mid (E) \mid id\}, E) \]
Derivations

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

- $\varphi \Rightarrow \psi$
  
  if there exist $\varphi_1, \varphi_2 \in (V_N \cup V_T)^*, A \in V_N$
  
  - $\varphi \equiv \varphi_1 A \varphi_2$
  
  - $A \rightarrow \alpha \in P$
  
  - $\psi \equiv \varphi_1 \alpha \varphi_2$

- $\varphi \Rightarrow^* \psi$ reflexive transitive closure

- The language defined by $G$

  $L(G) = \{ w \in V_T^* \mid S \Rightarrow^* w \}$
Reduced and Extended Context Free Grammars

A nonterminal $A$ is

reducible: There exist $\varphi_1, \varphi_2$ such that $S \Rightarrow^* \varphi_1 A \varphi_2$

productive: There exists $w \in V_T^*, A \Rightarrow^* w$

Removal of unreachable and non-productive nonterminals and the productions they occur in doesn't change the defined language. A grammar is reduced if it has neither unreachable nor non-productive nonterminals.

A grammar is extended if a new start symbol $S'$ 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 nonterminals.
- Leaves are labeled by terminals or by $\varepsilon$.
- For internal nodes $n$: If $n$ is labeled by $N$ and are its children $n.1, \ldots, n.n_p$ labeled by $N_1, \ldots, N_{n_p}$, then $N \rightarrow N_1, \ldots, N_{n_p} \in P$. 

$X \rightarrow X_1 \ldots X_n$
Examples
Leftmost (Rightmost) Derivations

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

- $\varphi \xrightarrow{lm} \psi$ 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 nonterminal

- $\varphi \xrightarrow{rm} \psi$ 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 nonterminal

- $\varphi \xrightarrow{lm} \psi$, $\varphi \xrightarrow{rm} \psi$ 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.
Context-free language

Non-Autom. uneambig. C++

Deterministically analyzable

context-free language