Syntax Analysis

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

May 7, 1997

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” Compiler Structure

Phase	Technique
source (character string)	
↓	
lexical analysis(7)	finite automata
↓	
source (symbol string)	
↓	
syntax analysis(8)	pushdown automata
↓	
syntax-tree	
↓	
semantic-analysis(9)	attribute grammar evaluators
↓	
decorated syntax-tree	
↓	
optimizations(10)	abstract interpretation + transformations
↓	
intermediate rep.	
↓	
code-generation(11, 12)	tree automata + dynamic programming + $\cdots$
↓	
machine-program	

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

- Property of a parsing method

- All the parsing methods treated, i.e. LL-parsing and LR-parsing, have the valid prefix property.

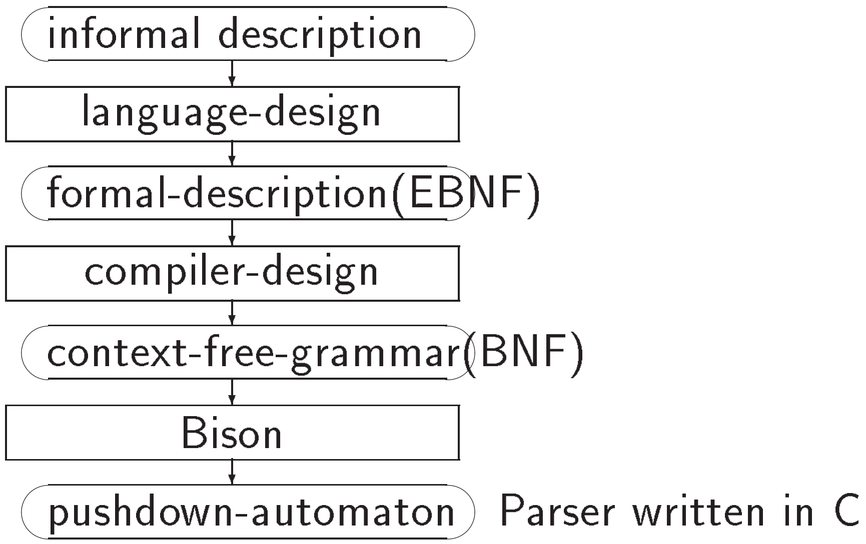
Error Diagnosis Data

- 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

The General Picture



Example Context Free Grammar (Section)

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**
— 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**

Examples

$$
\begin{array}{l}
G_0 = (\{E, T, F\}, \{+, *, (,), \mathbf{id}\}, \\
\quad \{ \quad E \quad \rightarrow \; E + T \; \mid \; T \\
\qquad\quad T \quad \rightarrow \; T * F \; \mid \; F \\
\qquad\quad F \quad \rightarrow \; (E) \; \mid \; \mathbf{id} \\
\quad \}, \\
E)
\end{array}
$$

$$
\begin{array}{l}
G_1 = \\
(\{E\}, \{+, *, (,), \mathbf{id}\}, \{E \;\rightarrow\; E{+}E \;\mid\; E{*}E \;\mid\; (E) \;\mid\; \mathbf{id}\}, E)
\end{array}
$$

Derivations

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

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

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

Reduced Context Free Grammars

Every 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.

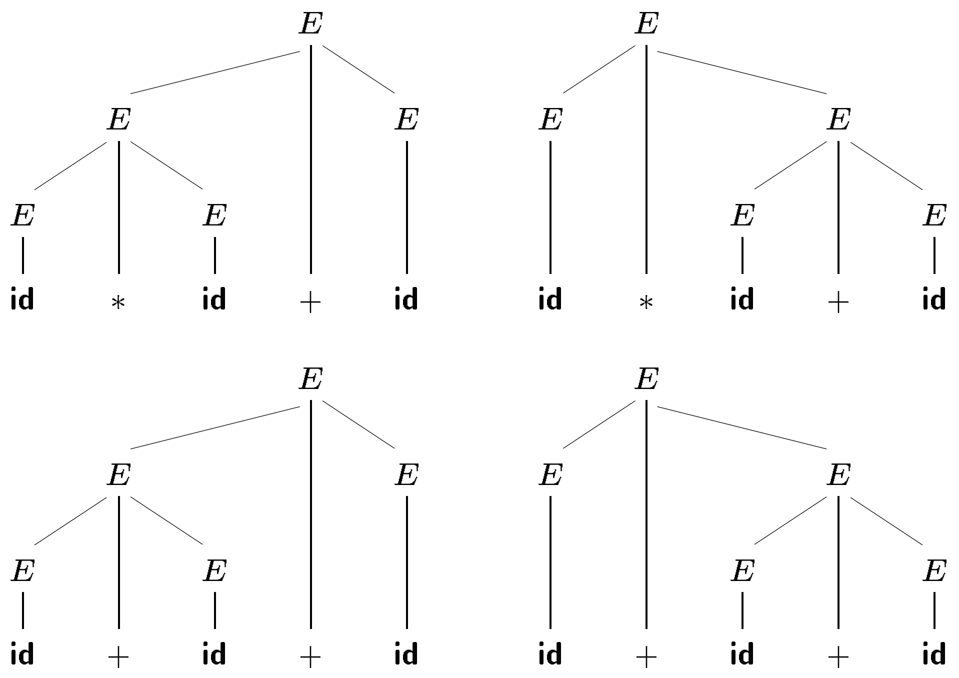
A grammar is **reduced** if it has neither unreachable nor unproductive non-terminals.

From now on, we only consider reduced 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 $n.1, \ldots, n.n_p$ labeled by $N_1, \ldots, N_{n_p}$, then $N \rightarrow N_1, \ldots, N_{n_p} \in P$.

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 which has (equivalently)

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