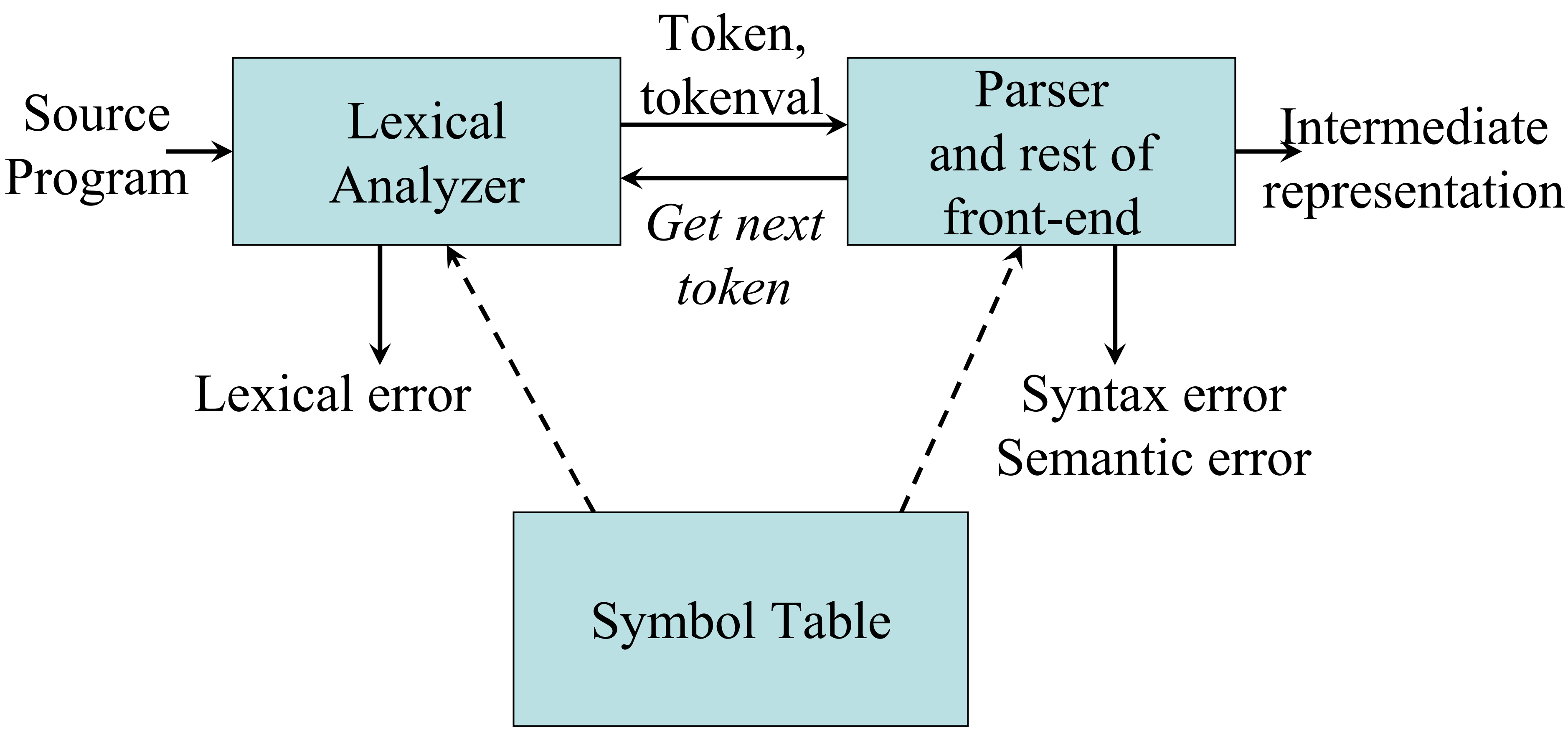
Syntax Analysis

Chapter 4

Position of a Parser in the Compiler Model



The Parser

- A parser implements a C-F grammar
- The role of the parser is twofold:

1. To check syntax (= string recognizer)
   - And to report syntax errors accurately
2. To invoke semantic actions
   - For static semantics checking, e.g. type checking of expressions, functions, etc.
   - For syntax-directed translation of the source code to an intermediate representation

Syntax-Directed Translation

- One of the major roles of the parser is to produce an intermediate representation (IR) of the source program using syntax-directed translation methods
- Possible IR output:
  - Abstract syntax trees (ASTs)
  - Control-flow graphs (CFGs) with triples, three-address code, or register transfer list notation
  - WHIRL (SGI Pro64 compiler) has 5 IR levels!

Error Handling

- A good compiler should assist in identifying and locating errors
  - *Lexical errors*: important, compiler can easily recover and continue
  - *Syntax errors*: most important for compiler, can almost always recover
  - *Static semantic errors*: important, can sometimes recover
  - *Dynamic semantic errors*: hard or impossible to detect at compile time, runtime checks are required
  - *Logical errors*: hard or impossible to detect

Viable-Prefix Property

- The *viable-prefix property* of parsers allows early detection of syntax errors
  - Goal: detection of an error *as soon as possible* without further consuming unnecessary input
  - How: detect an error as soon as the prefix of the input does not match a prefix of any string in the language

Error is detected here

```
...
for (;)
...
```

Prefix

Error is detected here

```
...
DO 10 I = 1;0
...
```

Prefix

Error Recovery Strategies

- *Panic mode*
  - Discard input until a token in a set of designated synchronizing tokens is found
- *Phrase-level recovery*
  - Perform local correction on the input to repair the error
- *Error productions*
  - Augment grammar with productions for erroneous constructs
- *Global correction*
  - Choose a minimal sequence of changes to obtain a global least-cost correction

Grammars (Recap)

- Context-free grammar is a 4-tuple
  $G = (N, T, P, S)$ where
  - T is a finite set of tokens (*terminal* symbols)
  - N is a finite set of *nonterminals*
  - P is a finite set of *productions* of the form
    $\alpha \rightarrow \beta$
    where $\alpha \in (N \cup T)^* \; N \; (N \cup T)^*$ and $\beta \in (N \cup T)^*$
  - $S \in N$ is a designated *start symbol*

Notational Conventions Used

- Terminals
  $a,b,c,\ldots \in T$
  specific terminals: **0**, **1**, **id**, **+**
- Nonterminals
  $A,B,C,\ldots \in N$
  specific nonterminals: *expr*, *term*, *stmt*
- Grammar symbols
  $X,Y,Z \in (N \cup T)$
- Strings of terminals
  $u,v,w,x,y,z \in T^*$
- Strings of grammar symbols
  $\alpha,\beta,\gamma \in (N \cup T)^*$

Derivations (Recap)

- The *one-step derivation* is defined by
  $\alpha\, A\, \beta \Rightarrow \alpha\, \gamma\, \beta$
  where $A \rightarrow \gamma$ is a production in the grammar
- In addition, we define
  - $\Rightarrow$ is *leftmost* $\Rightarrow_{lm}$ if α does not contain a nonterminal
  - $\Rightarrow$ is *rightmost* $\Rightarrow_{rm}$ if β does not contain a nonterminal
  - Transitive closure $\Rightarrow^*$ (zero or more steps)
  - Positive closure $\Rightarrow^+$ (one or more steps)
- The *language generated by G* is defined by
  $L(G) = \{w \in T^* \mid S \Rightarrow^+ w\}$

Derivation (Example)

Grammar $G = (\{E\}, \{\mathbf{+,*,(,),-,id}\}, P, E)$ with productions $P =$

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E \mathbin{\mathbf{*}} E \\
E &\rightarrow \mathbf{(}\ E\ \mathbf{)} \\
E &\rightarrow \mathbf{-}\, E \\
E &\rightarrow \mathbf{id}
\end{aligned}
$$

Example derivations:

$E \Rightarrow \mathbf{-}\, E \Rightarrow \mathbf{-\ id}$

$E \Rightarrow_{rm} E + E \Rightarrow_{rm} E + \mathbf{id} \Rightarrow_{rm} \mathbf{id} + \mathbf{id}$

$E \Rightarrow^{*} E$

$E \Rightarrow^{*} \mathbf{id} + \mathbf{id}$

$E \Rightarrow^{+} \mathbf{id} \mathbin{\mathbf{*}} \mathbf{id} + \mathbf{id}$

Chomsky Hierarchy: Language Classification

- A grammar G is said to be
  - *Regular* if it is *right linear* where each production is of the form
    $A \rightarrow w\,B$ or $A \rightarrow w$
    or *left linear* where each production is of the form
    $A \rightarrow B\,w$ or $A \rightarrow w$
  - *Context free* if each production is of the form
    $A \rightarrow \alpha$
    where $A \in N$ and $\alpha \in (N \cup T)^*$
  - *Context sensitive* if each production is of the form
    $\alpha\, A\, \beta \rightarrow \alpha\, \gamma\, \beta$
    where $A \in N$, $\alpha,\gamma,\beta \in (N \cup T)^*$, $|\gamma| > 0$
  - *Unrestricted*

Chomsky Hierarchy

$\mathsf{L}(\textit{regular}) \subset \mathsf{L}(\textit{context free}) \subset \mathsf{L}(\textit{context sensitive}) \subset \mathsf{L}(\textit{unrestricted})$

Where $\mathsf{L}(T) = \{\ L(G) \mid G \text{ is of type } T\ \}$
That is: the set of all languages
generated by grammars G of type T

Examples:

Every *finite language* is regular! (construct a FSA for strings in $L(G)$)

$L_1 = \{\ \mathbf{a}^n\mathbf{b}^n \mid n \geq 1\ \}$ is context free

$L_2 = \{\ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n \geq 1\ \}$ is context sensitive

Parsing

- *Universal* (any C-F grammar)
  - Cocke-Younger-Kasimi
  - Earley
- *Top-down* (C-F grammar with restrictions)
  - Recursive descent (predictive parsing)
  - LL (Left-to-right, Leftmost derivation) methods
- *Bottom-up* (C-F grammar with restrictions)
  - Operator precedence parsing
  - LR (Left-to-right, Rightmost derivation) methods
    - SLR, canonical LR, LALR

Top-Down Parsing

- LL methods (Left-to-right, Leftmost derivation) and recursive-descent parsing

Grammar:

$E \rightarrow T + T$

$T \rightarrow \mathbf{(}\ E\ \mathbf{)}$

$T \rightarrow \mathbf{-}\ E$

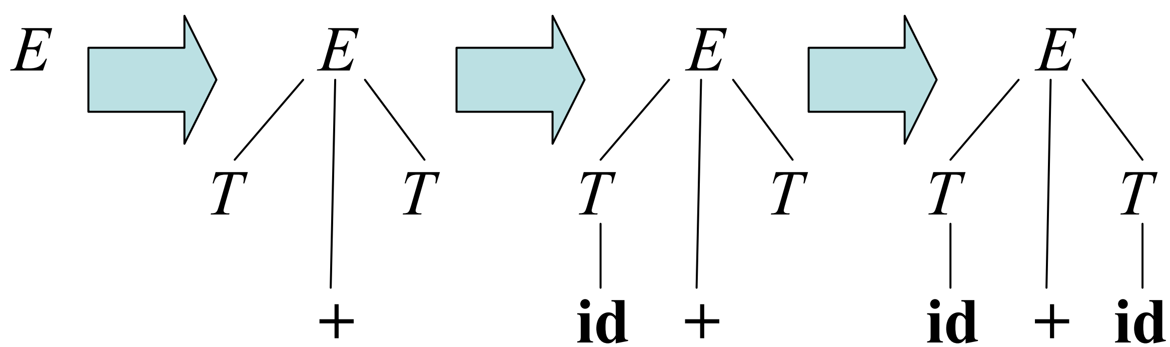
$T \rightarrow \mathbf{id}$

Leftmost derivation:

$E \Rightarrow_{lm} T + T$

$\Rightarrow_{lm} \mathbf{id} + T$

$\Rightarrow_{lm} \mathbf{id} + \mathbf{id}$



Left Recursion (Recap)

- Productions of the form

$$A \to A\ \alpha$$
$$\mid \beta$$
$$\mid \gamma$$

  are left recursive
- When one of the productions in a grammar is left recursive then a predictive parser loops forever on certain inputs

Immediate Left-Recursion Elimination

Rewrite every left-recursive production

$$
\begin{aligned}
A \rightarrow\ & A\ \alpha \\
\mid\ & \beta \\
\mid\ & \gamma \\
\mid\ & A\ \delta
\end{aligned}
$$

into a right-recursive production:

$$
\begin{aligned}
A \rightarrow\ & \beta\ A_R \\
\mid\ & \gamma\ A_R \\
A_R \rightarrow\ & \alpha\ A_R \\
\mid\ & \delta\ A_R \\
\mid\ & \varepsilon
\end{aligned}
$$

A General Systematic Left Recursion Elimination Method

Input: Grammar G with no cycles or ε-productions

Arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$

for $i = 1, \ldots, n$ **do**

  for $j = 1, \ldots, i$-1 **do**

    replace each

      $A_i \to A_j\, \gamma$

    with

      $A_i \to \delta_1\, \gamma \mid \delta_2\, \gamma \mid \ldots \mid \delta_k\, \gamma$

    where

      $A_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$

  enddo

  eliminate the *immediate left recursion* in A_i

enddo

Example Left Recursion Elim.

$$\left.\begin{array}{l} A \rightarrow B\ C \mid \mathbf{a} \\ B \rightarrow C\ A \mid A\ \mathbf{b} \\ C \rightarrow A\ B \mid C\ C \mid \mathbf{a} \end{array}\right\}$$ Choose arrangement: A, B, C

$i = 1$: nothing to do

$i = 2, j = 1$: $B \rightarrow C\ A \mid \underline{A}\ \mathbf{b}$

$\Rightarrow \quad B \rightarrow C\ A \mid \underline{B\ C}\ \mathbf{b} \mid \underline{\mathbf{a}}\ \mathbf{b}$

$\Rightarrow_{\text{(imm)}} \quad B \rightarrow C\ A\ B_R \mid \mathbf{a}\ \mathbf{b}\ B_R$

$\quad B_R \rightarrow C\ \mathbf{b}\ B_R \mid \varepsilon$

$i = 3, j = 1$: $C \rightarrow \underline{A}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{B\ C}\ B \mid \underline{\mathbf{a}}\ B \mid C\ C \mid \mathbf{a}$

$i = 3, j = 2$: $C \rightarrow \underline{B}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{C\ A\ B_R}\ C\ B \mid \underline{\mathbf{a}\ \mathbf{b}\ B_R}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow_{\text{(imm)}} \quad C \rightarrow \mathbf{a}\ \mathbf{b}\ B_R\ C\ B\ C_R \mid \mathbf{a}\ B\ C_R \mid \mathbf{a}\ C_R$

$\quad C_R \rightarrow A\ B_R\ C\ B\ C_R \mid C\ C_R \mid \varepsilon$