The role of the parsing process is to reconstruct a derivation of a given input string by a given Context Free Grammar. Equivalently, construct a parsing tree which represents a given input string. Parsing

Each recognizing function returns a tree fragment.
- Each terminal
  - expected terminal
  - lexicial scanner failure if the resulting token is not the terminal
- Each non-terminal in the R.H.S. is translated into a call to the function (procedure) that recognizes that non-terminal
- Each non-terminal on the R.H.S. is translated into a call to the body of the function
- Each right-hand side of a production provides part of the deriviation process.
- Each recognizing function returns a tree fragment.

Recursive Descent Parsing

General Structure

We consider first an ad-hoc manual method, called the derivation. Equivalently, construct the parsing tree which represents

The parsing process

The role of the parsing process is to reconstruct a derivation of a given input string by a given Context Free Grammar. Equivalently, construct a parsing tree which represents a given input string.
Example: Parsing a Declaration

FULL TYPE DECLARATION ::= type DEFINING IDENTIFIER is TYPE DEFINITION ;

Translates into

gettoken type
Find a defining-identifier — function call
get token is type-definition
Next token is type-definition
Translates into

FOR-TYPE-DEFINITION := "" ;

Example: Parsing a Loop

FOR STATEMENT ::= ITERATION SCHEME loop STATEMENTS endloop ;

Translates into

Node1 := functioncall gettoken loop
List1 := functioncall gettoken loop
gettoken semicolon
Result := build-loop-node with Node1 and List1
return Result

Example: Factor Grammar

If several productions have the same prefix, rewrite as a single production:

IF-STAT :: = if COND then Stats [ELSE PART] endif ;

Problem now reduces to recognizing whether an optional component (ELSE PART) is present.

Solution: Factor Grammar

Induction.
Collect functions returning a failure, this function returns a failure in case we fail to find any of the expected tokens or one of the

return result
result := build-loop-node with Node1 and List1
gettoken semicolon
gettoken loop
gettoken end loop
call functioncall
sequence-of-statements
gettoken semicolon
node1 := build-iteration-node with

Translation scheme loop statements end loop ;
Consider rule if cond then stats [else stats] end if;

The grammar:

Non-Terminals

Left Recursion Involving Several

Informally: $E \rightarrow + T E$ is a possibly empty sequence of terms

Original scheme leads to an infinite loop: grammar is inappropriate for recursive descent.

Problem: to find an $E$, start by finding an $T$...

Informally: $E \rightarrow T + T$ means that eventually $E$

End get-if

if token=end then
	return 0;
endif

if token=if then
	if get-stat()=0 then return 0;
	if get-cond()=0 then return 0;
	if get-stat()=1 then return 0;
	if get-stat()=1 then return 0;
	if get-stat()=1 then return 0;
	if get-stat()=1 then return 0;

boolean function get-if() =

Consider rule

Informally: $E \rightarrow + T E$ can be rewritten as

Rewrite as $E \rightarrow TE$ $E_0 \rightarrow + TE \leftrightarrow TE$ $E_0 \rightarrow + TE$

Informally: $E_0$ is a possibly empty sequence of terms ($T$) each preceded by a +.

Informally: $A \rightarrow BC$ $A \rightarrow BC$ $B \rightarrow AE$ $B \rightarrow AF$

Can be rewritten as

The grammar:

and then apply previous method

A | RC | AEC | $E_0$ $A$ | RC | $AEC$ | $E_0$
Lecture 3: Parsing
A. Pnueli

Honors Compilers, NYU, Fall, 2009

The General Case

Further Complications

Table-Driven Parsing

Honors Compilers, NYU, Fall, 2009
A PDA is defined by a tuple \((Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)\), where:

- **States** \(Q\) — A finite set of states
- **Input alphabet** \(\Sigma\) — The input alphabet
- **Stack alphabet** \(\Gamma\) — The stack alphabet
- **Initial state** \(q_0 \in Q\) — The initial state
- **Initial stack symbol** \(Z_0 \in \Gamma\) — The initial stack symbol
- **Set of accepting states** \(F \subseteq Q\) — The set of accepting states

**Formally**

The transition function \(\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)\) is a non-deterministic transition function.

**Runs and Acceptance**

An instantaneous description (ID) is an \(A\)-successor of the ID \((p; ax; X)\), written \((p; ax; X) \xrightarrow{\delta} (q; x; \alpha)\), if \((q, x, \alpha) \in \delta(p, a, X)\).

An alternate definition is provided by the notion of the word accepted by an empty stack.

**PDA’s Can Accept Languages Beyond FSM’s**

For example, the language of balanced parentheses expressions. This language can be generated by the following grammar:

\[
S ::= () \mid (S) \mid SS
\]

A PDA which accepts this language is given as follows:

\[
\delta(q_0, \epsilon, Z_0) = (q_1, Z_0, Z_0) = (q_2, \epsilon)
\]

The transition function for this automaton can be given by:

\[
\delta(q_0, Z_0, Z_0) = (q_1, Z_0, Z_0) = (q_2, \epsilon)
\]
A PDA is defined to be deterministic (DPDA) if it has no -moves, and for every \( b \in \Sigma \), \( a \in \Sigma \) and \( X \in \Sigma \) and \( X \not\in I \),

\[
\{(I < \ell | \sigma_{q,0}) \} \cap \{(I < \ell | \sigma_{q,0}) \} 
\]

DPDA (deterministic PDA). For example, there exist CFLs which cannot be recognized by a DPDA.

A language \( I \) is recognized by a (possibly non-deterministic) PDA if it is a CFL (can be generated by a CFG) and for every \( b \in \Sigma \), \( a \in \Sigma \) and \( X \in \Sigma \) and \( X \not\in I \),

\[
|X, a, b) \} 
\]

no -moves

A PDA is defined to be deterministic (DPDA) if it has no -moves.

Semantic Action

When choosing a production, build the node for non-terminal, attach to parent \( T \).

If stack and input string are both empty, apply the accept.

Do not consume \( a \).

Otherwise, choose a grammar production \( T \rightarrow \alpha \). Replace \( T \) by \( \alpha \) and consume input. This is called a match action.

If \( T \) is a terminal symbol, then \( T \) must equal \( a \). Replace stack and the next input token.

At each step, let \( T \) be the symbol at top of the stack and \( a \) be the next input token.

Initially, stack contains Grammar start symbol \( S \).

\[
\text{Stack} \quad \text{Input} \quad \text{Action}
\]

\[
\begin{align*}
(\_)(\_)
\rightarrow (s)(\_)
\rightarrow (ss)
\rightarrow (s)
\rightarrow s
\end{align*}
\]
We Need Deterministic Parsing

Claim 5. Every PD is equivalent to a single-state PDA.

What about multi-state PDAs?

Claim 6. No PDA accepts an empty stack.

Proof: Let A be a PD accepting a language L. Then no PD accepts a language L which includes \{e\}.

Correspondence of PDAs to CFGs

A CFG corresponds to a PD if

For every CFG G, there exists a PD A which accepts by top-down parsing the language L(G).

For every PD A, there exists a CFG G which accepts by top-down parsing the language L(A).

For every PD A, there exists a CFG G which accepts by leftmost derivation the string x.

For every CFG G, there exists a PD A which accepts by leftmost derivation the string x.

The transition function is defined as the smallest relation satisfying

Acceptance is by empty stack.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.

The grammar start symbol Z0 is a single-state PDA.
The following grammar for balanced parentheses expressions is not $LL(k)$ for any $k$:

\[
S \::= ( | (S) | SS
\]

A 2-lookahead is sufficient in order to distinguish between $( )$ and $(S)$. However, no bounded lookahead is sufficient in order to distinguish between $(S)$ and $SS$.

The following grammar is $LL(1)$:

\[
S \::= ( | (S) | SS
\]

If the next input character is $\epsilon$ we choose $(S)S$. Otherwise we choose $\epsilon$.

**Computing First($X$)**

- If $X$ is terminal, then $\text{First}(X) = \{ X \}$.
- For each non-terminal $X$ and production $X \rightarrow \alpha_1 \cdots \alpha_k$:
  - Add $\alpha_i$ to $\text{First}(X)$ if, for some $i$, $\alpha_i \in \text{First}(\alpha_j)$; for all $j \in [i+1,k]$.
  - Add $\epsilon$ to $\text{First}(X)$ if $\alpha_i \notin \text{First}(\alpha_j)$ for all $j \in [i+1,k]$.

**Examples**

- Let $k > 0$ be a positive integer. The grammar $G$ is called an $LL(k)$ grammar if, for every leftmost derivation $A \rightarrow_1 \beta_1 \cdots \beta_k \rightarrow_1 xy$, the production $A \rightarrow \alpha_1 \cdots \alpha_k$ is uniquely determined by $A$, and the first $k$ characters of $y$.

- The name is based on the fact that parsing according to such a grammar reads the input from left to right while constructing a leftmost derivation with a lookahead of $k$ characters.

- The unique determination means that we have two derivations of the form $S \rightarrow_1 x_1 \beta_1 \alpha_1 \rightarrow_1 x_1 \beta_1 \beta_2 \alpha_2 \rightarrow_1 x_1 \beta_1 \beta_2 \beta_3 \alpha_3 \rightarrow_1 \cdots \rightarrow_1 x_1 \beta_1 \beta_2 \cdots \beta_k \alpha_k$, then $\beta_1 = \beta_2 = \cdots = \beta_k$.

**Constructing $LL(1)$ Tables**

- Define two functions on the symbols of the grammar:
  - First and Follow.
- For a non-terminal $A \in N$, $\text{First}(A)$ is the set of terminals that can appear as the first character in a string derived from $A$.
- For a string $X_1 \cdots X_k$, and terminal $a$, we say that $a \in \text{First}(X)$ if and only if $a \in \text{First}(X_1) \cap \cdots \cap \text{First}(X_k)$.
- $\text{Follow}(A) = \{ \epsilon \in T \mid S \rightarrow \epsilon \Rightarrow * \}
- For a string $X_1 \cdots X_k$, and terminal $a$, we say that $a \in \text{Follow}(X)$ if and only if $a \in \text{First}(X_2) \cap \cdots \cap \text{First}(X_k)$ for some $i \in [1,k]$.

- $\text{Follow}(A) = \{ \epsilon \in T \mid S \rightarrow \epsilon \Rightarrow * \}
- For a string $X_1 \cdots X_k$, and terminal $a$, we say that $a \in \text{Follow}(X)$ if and only if $a \in \text{First}(X_2) \cap \cdots \cap \text{First}(X_k)$ for some $i \in [1,k]$.
leading to the following parsing table:

```
First/Follow

Non-Terminal   FIRST   FOLLOW

E              
T              
L              

<table>
<thead>
<tr>
<th>Input</th>
<th>Action</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>$L</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>$L*</td>
</tr>
<tr>
<td>$L*</td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>$L</td>
</tr>
<tr>
<td>$L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

We can parse the following inputs:

```
Parsing of Arithmetic Expressions

Example: Parsing Table 1
```

This is the parsing table for the grammar:

```
A       B       C

A       B

A       B

A       B

A       B
```

Such that \( \beta \in \text{Follow}(A) \) and \( A \rightarrow \alpha \) to \( \alpha \rightarrow A \) if \( \alpha \in \text{First}(A) \), then for each terminal \( \alpha \in \text{First}(A) \), add \( \alpha \rightarrow A \), where \( \alpha \) is terminal or non-terminal.

If there is a production \( A \rightarrow B \) and all symbols in \( B \) are in \( \text{Follow}(A) \) \( \beta \), where \( \beta \in \text{First}(A) \), then add all symbols in \( \{ \beta \} \) to \( \text{Follow}(B) \). If there is a production \( A \rightarrow B \) and all symbols in \( B \) are in \( \text{Follow}(A) \) \( \beta \), then add all symbols in \( \{ \beta \} \) to \( \text{Follow}(B) \). If there is a production \( A \rightarrow B \) and all symbols in \( B \) are in \( \text{Follow}(A) \) \( \beta \), then add all symbols in \( \{ \beta \} \) to \( \text{Follow}(B) \). If there is a production \( A \rightarrow B \) and all symbols in \( B \) are in \( \text{Follow}(A) \) \( \beta \), then add all symbols in \( \{ \beta \} \) to \( \text{Follow}(B) \).
Correctness of the Construction

Claim 6. A grammar \( G \) is an \( LL(1) \) grammar if the parsing table \( M[A,a] \) contains at most one production in each entry.