Parsing
The role of the parsing process is to reconstruct a derivation by which a given context-free grammar can generate a given input string. Equivalently, construct the parsing tree which represents the derivation.

We consider first an ad-hoc manual method, called Recursive Descent which attempts to emulate the derivation process.
Recursive Descent Parsing

Each procedure recognizes an instance of a non-terminal and returns tree fragment for the non-terminal.

Each production corresponds to one recursive procedure.

Top-down parsing builds tree from root symbol.
Each recognizing function returns a tree fragment.

- Each terminal in the rhs is translated into a call to the recognizing function (procedure) that recognizes that non-terminal expected terminal.

- Error if the resulting token is not the lexical scanner.

- Each right-hand side of a production provides a body.

General Structure
Example: Parsing a Declaration

`FULL.DECLARATION ::= type DEFINING IDENTIFIER is TYPE-DEFINITION;`
Example: Parsing a Loop

Let's consider the following program:

```plaintext
FOR STATEMENT ::= ITERATION SCHEME loop STATEMENTS endloop;
```

Translates into:

```plaintext
RESULT ::= build loop node with Node1 and List1
RESULT ::= Sequence of statements
RESULT ::= build loop semicolon
RESULT ::= build loop
RESULT ::= Sequence of statements
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::= build loop
RESULT ::
If there are multiple productions for a non-terminal, we need a mechanism to determine which production to use. When next token is `if`, cannot tell which production to use.

```
IF-STAT $::= if COND then Stats ELSE PART end if;
```

Complcations
If several productions have the same prefix, rewrite as:

IF-STAT \[ ::= \]

\begin{align*}
  & \text{if} \ \text{COND} \ \\
  & \text{then} \ \\
  & \text{Stats[ELSE-PART]} \\
  & \text{endif;}
\end{align*}

Problem now reduces to recognize whether an optional component (ELSE-PART) is present. With a single token lookahead this is possible — look for a token else.

Else.

Solution: Factor Grammar
Consider rule •

Illustrate Solution

A. Pnueli

Honors Compilers, NYU, Spring, 2007

Lecture 3: Parsing
Complication: Left Recursion

Grammar cannot be left-recursive.

Original scheme leads to an infinite loop: grammar is

\[ L \mid L + E =:: E \]

Problem: to find an \( E \), start by finding an \( E \).

Inappropriate for recursive descent.
(\ell) each preceded by +. 

Informally: \( E \) is a possibly empty sequence of terms.

\[
E \quad | \\quad \ell E + \ell E \\
\ell E \\
\ell E =:\quad E
\]

Rewrite as

\[
\ldots \ell + \ell + \ell
\]

Expands into

\( E \) means that eventually \( \ell \quad | \\quad \ell + E =:\quad E \)

Eliminating Left Recursion
and then apply previous method

\[ A ::= BC \mid \text{ } D \]

\[ B ::= AE \mid FC \]

\[ C ::= \text{ } AE \mid \text{ } FC \]

\[ D ::= \text{ } BC \]

Can be rewritten as

\[ A ::= AE \mid FC \]

\[ B ::= AE \]

\[ C ::= \text{ } AE \mid FC \]

\[ D ::= BC \]

The grammar

- Non-Terminals
- Left Recursion Involving Several Non-Terminals
Arrange the non-terminals in some order $A_1, \ldots, A_n$.

The General Case

Eliminate the immediate left recursion among $A$-productions:

Replace each production of the form $A_i \rightarrow A_j$ by the productions $A_i \rightarrow 1 \in_{j\cap i}$, where $A_j \rightarrow 1 \in_{j\cap i}$ are all current $A_j$-productions.

for $j \in [1, \ldots, i-1]$ do

for $j \in [n, \ldots, i]$ do

for $\in [1, \ldots, i]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ldots, n]$ do

for $\in [1, \ld···}
In practice, treat as $E$ 

Incorrect for $a-b-c$: must rewrite tree.

$E ::= T E T E$ Parses $a+b+c$ as $a+(b+c)$

$E ::= E + E$ Parses $a+b+c$ as $(a+b)+c$

Transformation does not preserve associativity.

Further Complications
Use loop to Parse Sequence of Terms

- Find next term
  - operand for next operation

- past operator
  - Scan

end loop;

Node1 := Node2;
Node2.right := P-term;
Node2.left := Node1;

Node2.op := Token;
Node2 := New-Node(P-Binary-Addding-Operator);
exit when Token not in Token-Class-Binary-Addding;

loop

Node1 := P-term;
call function that parses a term

A. Pnueli

Honors Compilers, NYU, Spring, 2007
Table-Driven Parsing

FSM driven by (table(s) generated) automatically from grammar

Parsings performed by a finite-state machine augmented

Tables Generator Language
A context-free language can be recognized by a finite state machine with a stack: a \textbf{PDA}.

Pushdown Automata

Acceptance can be defined by either accepting state or reaching an empty stack.

- Transition table
- The PDA is defined by set of internal states and a stack:
  \textbf{PDA}.

The PDA can read the input and read/write to the top of the stack.

Actions of the PDA are determined by the current state, the current symbol at the top of the stack and the current input character.

A PDA is a finite state machine with a stack:

- Transition table
- Current input character
- The PDA can be defined by a set of internal states and a stack:
  \textbf{PDA}.

- Accepting state or emptystack
A PDA is defined by a tuple \( \langle \mathcal{M}, \mathcal{O}, Z_0, Q_0, \mathcal{I}, \mathcal{F} \rangle \), where:

- \( \mathcal{M} \) — A non-deterministic transition function
- \( \mathcal{O} \) — A finite set of states
- \( \mathcal{I} \) — The input alphabet
- \( \mathcal{Z} \) — The stack alphabet
- \( \mathcal{F} \) — The set of accepting states
- \( Z_0 \) — The initial stack symbol
- \( Q_0 \) — The initial state

A PDA is formally defined by:

- \( \mathcal{M} = (\mathcal{O} \times \mathcal{O})^* \mathcal{I} \times \mathcal{O} \leftarrow \mathcal{I} \times (\{\varepsilon\} \cup \mathcal{Z}) \times \mathcal{O} : \delta \)
A PDA which accepts this language is given as follows:

\[
\begin{align*}
(\epsilon, \epsilon) & = (Z, \epsilon, \epsilon, \epsilon) \\
(X \epsilon, \epsilon) & = (X, \epsilon, \epsilon, \epsilon) \\
(0Z0, \epsilon) & = (0Z, \epsilon, \epsilon, \epsilon)
\end{align*}
\]

The transition function for this automaton can be given by

\[
\begin{align*}
\delta(q_0; \epsilon, Z) &= (q_1, Z, X) \\
\delta(q_1; 0, X) &= (q_1, X, Z) \\
\delta(q_1; 0, Z) &= (q_1, Z, \epsilon) \\
\delta(q_1; 0, 0) &= (q_1, \epsilon, \epsilon)
\end{align*}
\]

For example, the language of balanced parentheses expressions.

\textbf{PDAs Can Accept Languages Beyond FSMs}
An instantaneous description \((q; x; e)\) is a tuple \((q, x, e)\), where \(q\) is a state, \(x\) is the input left to read, and \(e\) is the current stack contents.

For some \(m \in \mathbb{N}\) the word \(w \in \Sigma^* \) is accepted by \(A\) if \((q_0; w; Z_0) \rightarrow^* (q; \varepsilon, Z)\) for some \(q \in F\) and \(Z \in \Sigma^* \).

The language recognized by \(A\) is the set of all words accepted by \(A\).

\(\forall A \in \mathcal{A}\), \(\forall I \in \mathcal{I}\), \(\forall m \in \mathbb{N}\), the reflexive-transitive closure of \(\rightarrow\) is defined by the rules:

\[ \left( (X', a', d) \mathrel{\vDash} \left( X, a, d \right) \mathrel{\rightarrow} \left( (g', b) \mathrel{\vDash} \left( g, b \right) \right) \right) \]  

For a PDA \(A\) the ID is an A-successor of the ID.

A PDA \(A\) accepts by an empty stack if for some \(m \in \mathbb{N}\) the notion of the word \(m\) is provided by the notion of the word \(m\).
Lecture 3: Parsing

A.Pnueli Properties of PDA’s and CFL’s

A PDA (deterministic PDA). For example,

\[
\{ i < j | p_i \geq q_j \} \cap \{ i < j | c \geq q_j \}
\]

There exist CFL’s which cannot be recognized by a

PDA.

A language is a CFL (can be generated by a CFG) if it is recognizable by a (possibly non-deterministic PDA).

A PDA is defined to be deterministic (DPDA) if it has no \( \varepsilon \)-moves, and for every \( a \in \Sigma \), \( X \in \Gamma \), and \( b \in \Sigma \), for every \( q \in Q \), \( \Delta \) has
Top-Down Parsing

A. Pnueli

1. Parsing tree is synthesized from the root (sentence).
2. Stack contains symbols of the parse tree from the recent production and pending non-terminals.
3. Transition table indexed by stack's top symbol and input symbol $a$.
4. Automaton is trivial (no need for explicit states).
5. Stack contains symbols of RHS of most recent production or (set of RHS symbols of terminal or).
6. Entries in table are terminals or (set of RHS symbols of terminal or).
Actions

Top-Down Parsing

Semantic Action when choosing a production, build tree node

• Initially, stack contains Grammar start symbol $S$.

• At each step, let $a$ be the symbol at top of the stack and $S$ be

• Semantic Action: when choosing a production, build tree node

• If stack and input string are both empty, then accept.

• Otherwise, choose a grammar production $T \rightarrow \alpha$. Replace by $L\alpha \leftarrow L$. Pop stack and do not consume $a$.

• If stack is a terminal symbol, then must equal $a$. Pop stack and consume input. This is called a match action.

• For non-terminal, attach to parent $L$. Honors Compilers, NYU, Spring, 2007.
Consider the grammar

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

and the leftmost derivation

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

A top-down recognition by a PDA is given by:

<table>
<thead>
<tr>
<th>Input</th>
<th>Stack</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ):: S</td>
<td>Expand</td>
<td>( )</td>
</tr>
<tr>
<td>( ):: S</td>
<td>Match</td>
<td>()()</td>
</tr>
<tr>
<td>SS:: S</td>
<td>Expand</td>
<td>()()</td>
</tr>
<tr>
<td>(S):: S</td>
<td>Expand</td>
<td>()()</td>
</tr>
<tr>
<td>(S):: S</td>
<td>Accept</td>
<td>()()</td>
</tr>
</tbody>
</table>

**Example: Top-Down Parsing**
A PDA corresponding to a CFG.

Claim 3. For every CFG $G$ there exists an automaton run $\delta$ such that:

$$\delta(0,0) : \langle G, 0^* \epsilon, 0^* \epsilon, 0^* \epsilon \rangle : A$$

where $A$ is the language corresponding to the PDA $\delta$.
Claim 5. Every PDA is equivalent to a single-state PDA.

What about multi-state PDAs?

Empty stack, then empty stack is accepted by $A$ if the string $w$ is accepted by $A$.

It follows that for each production set $P$, there exists a leftmost derivation of the form $\langle X, a \rangle$ whenever $w$ is accepted by $A$ with empty stack.

We can show by induction on the length of the run that, whenever $\langle X, a \rangle \in (\& \& b)$, we can construct a single-state PDA $A'$ for each production set $P$ containing the rule $P, X \mapsto b$.

Let $A$ be a single-state PDA which generates the language recognized by $A$ with empty stack.

For every single-state PDA $A$, there exists a CFG $G$ which...
Consider the PDA $A$:

\begin{align*}
\delta(q_0, \epsilon, \epsilon) &= \{q_1, q_2\} \\
\delta(q_1, b, \epsilon) &= \{q_2, q_3\} \\
\delta(q_2, \epsilon, \epsilon) &= \{q_1\} \\
\delta(q_3, \epsilon, \epsilon) &= \{q_2\}
\end{align*}

where the transition function is given by the following table:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$\epsilon$</th>
<th>$\emptyset$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$\emptyset$</td>
<td>$S \rightarrow S$</td>
<td>$S$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$(L)$</td>
<td>$L \in X$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$(L)$</td>
<td>$L \in X$</td>
</tr>
</tbody>
</table>

Following the recipe of Claim 4, we obtain the grammar $G$:

\begin{align*}
G &= \{ S \rightarrow S \epsilon, T \epsilon \} \\
N &= \{ S, T \} \\
\Sigma &= \{ \epsilon, \{ (, , ) \} \} \\
\Gamma &= \{ \emptyset, L \}
\end{align*}

Example: Converting a PDA to a CFG
When doing parsing, we are interested only in a parsing process which is based on a deterministic process. When doing parsing, we are interested only in a parsing process which is based on a deterministic process. We need deterministic parsing.

The results presented so far allowed non-deterministic general PDA’s. Unlike finite-state automata, deterministic pushdown automata (PDA) are strictly less expressive than general PDA’s. The context-free language \( \{ p_u z q_u v \} \cup \{ c_u q_u p \} \) cannot be recognized by a DPDA.
Let $k > 0$ be a positive integer. The grammar $G$ is called an LL($k$) grammar if, for every leftmost derivation $S \Rightarrow \cdots \Rightarrow \alpha \beta \gamma S$, the production $\alpha \Rightarrow \beta \gamma \beta$ is uniquely determined by $\alpha$ and the first $k$ characters of $\gamma$. The unique determination means that if we have two derivations of the form

\[
S \Rightarrow \cdots \Rightarrow \alpha_1 A_1 \Rightarrow \cdots \Rightarrow \alpha_n A_n \Rightarrow \cdots \Rightarrow \alpha_k x \Rightarrow \cdots \Rightarrow \gamma_1 \cdots \gamma_m \Rightarrow \cdots \Rightarrow y S
\]

such that $[\gamma_1 \cdots \gamma_m]_2 \Rightarrow \cdots \Rightarrow S$, then $[\alpha_1 \cdots \alpha_n]_2 \Rightarrow \cdots \Rightarrow S$. 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.
Examples

The following grammar is not $\text{LL}(k)$ for any $k$: $\text{LL}(1)$:

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

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

The following grammar is $\text{LL}(1)$:

\[
S \::= ()
\]

\[
S \::= (S) | () =:: S
\]
Denetwofunctionsonthesymbolsofthegrammar: \textsc{FIRST}and \textsc{FOLLOW}.

\textsc{FIRST}(A) = \{ \epsilon \mid A \in S \land \epsilon \in \text{FIRST}(N) \}

\textsc{FOLLOW}(A) = \{ \text{terminals that can appear after a string derived from } A \}

\{ \epsilon \mid A \in S \} \cup \{ \text{terminals that can appear as the first character in a string derived from } N \}

For a non-terminal \( A \in N \), \textsc{FIRST}(A), \textsc{FOLLOW}(A) are defined.

Constructing \textsc{LL(1)} Tables.
If $X$ is terminal, then $\text{FIRST}(X) = \{X\}$.

For each non-terminal $X$ and production $X \rightarrow Y_1 \cdots Y_k$, we say that $X$ is a production, add $\epsilon$ to $\text{FIRST}(X)$.

If $X \rightarrow \epsilon$ is a production, add $\epsilon$ to $\text{FIRST}(X)$.

For a string $X_1 \ldots X_k$ and terminal $a$, we say that $a \in \text{FIRST}(X_1 \ldots X_k)$ if $a \in \text{FIRST}(X_i)$, for some $i \in \mathbb{N}$, and $X_j \epsilon \text{FIRST}(X_j)$, for all $j \in \mathbb{N}$.

Also, $a \in \text{FIRST}(X_1 \ldots X_k)$ if $a \in \text{FIRST}(X_i)$, for all $i \in \mathbb{N}$.

For some terminal $a$, we say that $a \in \text{FIRST}(X_1 \ldots X_k)$ if $a \in \text{FIRST}(X_i)$, for all $i \in \mathbb{N}$.

Computing $\text{FIRST}(X)$.
Place $ in $FOLLOW(S)$, where $S$ is the start symbol.

If there is a production $A \rightarrow \epsilon$, then add all symbols in $FIRST(B)$ to $FOLLOW(B)$.

If there is a production $A \rightarrow aB$, then add all symbols in $FOLLOW(B)$ to $FOLLOW(A)$.

In $FOLLOW(A)$ to $FOLLOW(B)$, where $\epsilon \in FIRST(B)$, then add all symbols.

A Pnueli
Constructing an LL(1) Parsing Table

This is a table which, for each non-terminal $A$, tells us which non-terminal $L \in N \in A$ and next input character $a \in \Sigma$ should be taken. It helps us determine which production should next be taken.

For each production $A \rightarrow \alpha$ of the grammar, do the following:

- For each terminal $a \in \text{FIRST}(A)$, add $A \rightarrow \alpha$ to $M[A][a]$.
- If $a \in \text{FIRST}(A)$ and $b \in \text{FOLLOW}(A)$, then add $A \rightarrow \alpha$ to $M[A][b]$.
- For each terminal $b \in \text{FOLLOW}(A)$, add $A \rightarrow \alpha$ to $M[A][b]$ as well.

As well, if $a \in \text{FIRST}(A)$ and $b \in \text{FOLLOW}(A)$, then add $A \rightarrow \alpha$ to $M[A][a]$.
Example: Constructing $\text{LL}(1)$ Tables

### First/Follow Tables

<table>
<thead>
<tr>
<th>Non-Terminal</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>T</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Leading to the Following Parsing Table:

<table>
<thead>
<tr>
<th>Terminal</th>
<th>FIRST/FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^1$</td>
<td>$^2$</td>
</tr>
<tr>
<td>$^1$</td>
<td>$^3$</td>
</tr>
<tr>
<td>$^1$</td>
<td>$^4$</td>
</tr>
<tr>
<td>$^1$</td>
<td>$^5$</td>
</tr>
<tr>
<td>$^1$</td>
<td>$^6$</td>
</tr>
</tbody>
</table>

we construct the FIRST/FOLLOW tables:

- $e \rightarrow (E) \rightarrow e$
- $e \rightarrow \epsilon$
- $e \rightarrow \epsilon$
- $e \rightarrow \epsilon$
- $e \rightarrow \epsilon$

Starting with the grammar

A. Pnueli

Lecture 3: Parsing

Honors Compilers, NYU, Spring, 2007
We can parse with the following table:

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>id</td>
<td>$E'\rightarrow id$</td>
</tr>
<tr>
<td>$E'$</td>
<td>$E'$</td>
<td>$E'\rightarrow \epsilon$</td>
</tr>
<tr>
<td>$T'$</td>
<td>$T'$</td>
<td>$T'\rightarrow \epsilon$</td>
</tr>
<tr>
<td>$F'$</td>
<td>$F'$</td>
<td>$F'\rightarrow \epsilon$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>$id$</td>
<td>$TE'\rightarrow +TE'$</td>
</tr>
</tbody>
</table>
| $TE'
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.