CSC 4181 Compiler Construction

Context-Free Grammars

*Using grammars in parsers*

---

**Parsing Process**

- Call the scanner to get tokens
- Build a parse tree from the stream of tokens
  - A parse tree shows the syntactic structure of the source program.
- Add information about identifiers in the symbol table
- Report error, when found, and recover from the error

---

**Context-Free Grammar**

- a tuple \((V, T, P, S)\)
  - \(V\) is a finite set of nonterminals, containing \(S\)
  - \(T\) is a finite set of terminals,
  - \(P\) is a set of production rules in the form of \(\alpha \rightarrow \beta\), where \(\alpha\) is in \(V\) and \(\beta\) is in \((V \cup T)^*\), and
  - \(S\) is the start symbol.

- Any string in \((V \cup T)^*\) is called a *sentential form*.

---
Examples:

\[
E \to E \circ E \quad S \to SS \\
E \to (E) \quad S \to (S)S \\
E \to \text{id} \quad S \to () \\
O \to + \quad S \to \lambda \\
O \to - \\
O \to * \\
O \to / 
\]

Backus-Naur Form (BNF)

- Nonterminals are in < >.
- Terminals are any other symbols.
- ::= means \( \to \).
- | means or.
- Examples:

\[
\begin{align*}
<E> & ::= <E><O><E> | (<E>) | \text{ID} \\
<O> & ::= + | - | * | / \\
<S> & ::= <S><S> | (<S>)<S> | () | \lambda 
\end{align*}
\]

Derivation

- A sequence of replacement of a substring in a sentential form.

**Definition**

- Let \( G = (V, T, P, S) \) be a CFG, \( \alpha, \beta, \gamma \) be strings in \((V \cup \Sigma)^*\) and \( A \) is in \( V \).

\( \alpha\beta \Rightarrow_G \alpha\gamma\beta \) if \( A \to \gamma \) is in \( P \).

\( \Rightarrow_G \) denotes a derivation in zero step or more.
Examples

\[ S \rightarrow SS \mid (S)S \mid () \mid \lambda \]

\[ S \]
\[ \rightarrow SS \]
\[ \Rightarrow (S)S \]
\[ \Rightarrow (S)(S)S \]
\[ \Rightarrow (S)S(()S) \]
\[ \Rightarrow ((S)S)S((S)S) \]
\[ \Rightarrow ((S)S)((S)S)S \]
\[ \Rightarrow ((S)S)(S)S((S)S) \]
\[ \Rightarrow ((S)S)(S)((S)S)S \]
\[ \Rightarrow ((S)S)((S)S)((S)S)S \]
\[ \Rightarrow (((S)S)S)((S)S)((S)S)S \]

\[ E \rightarrow E \circ E \mid (E) \mid id \]

\[ O \rightarrow + \mid - \mid \ast \mid / \]

\[ E \rightarrow E \circ E \]
\[ \Rightarrow (E) \circ E \]
\[ \Rightarrow (E \circ E) \circ E \]
\[ \Rightarrow ((E \circ E) \circ E) \circ E \]
\[ \Rightarrow (((E \circ E) \circ E) \circ E) \circ E \]
\[ \Rightarrow (((id \circ E)) \circ E) \circ E \]
\[ \Rightarrow (((id + E)) \circ E) \circ E \]
\[ \Rightarrow (((id + id)) \circ E) \circ E \]
\[ \Rightarrow (((id + id)) \circ id) \circ E \]

Leftmost Derivation

- Each step of the derivation is a replacement of the leftmost nonterminals in a sentential form.

\[ E \]
\[ \rightarrow E \circ E \]
\[ \Rightarrow (E) \circ E \]
\[ \Rightarrow (E \circ E) \circ E \]
\[ \Rightarrow ((E \circ E) \circ E) \circ E \]
\[ \Rightarrow (((E \circ E) \circ E) \circ E) \circ E \]

Rightmost Derivation

- Each step of the derivation is a replacement of the rightmost nonterminals in a sentential form.

\[ E \]
\[ \rightarrow E \circ E \]
\[ \Rightarrow E \circ id \]
\[ \Rightarrow E * id \]
\[ \Rightarrow (E * id) * id \]
\[ \Rightarrow (id + id) * id \]

Right/Left Recursive

- A grammar is a left recursive if its production rules can generate a derivation of the form \( A \Rightarrow^* A \ X \).

- Examples:
  - \( E \rightarrow E \circ id \mid (E) \mid id \)
  - \( E \rightarrow F + id \mid (E) \mid id \)
  - \( F \rightarrow E \ast id \mid id \)
  - \( E \rightarrow F + id \)
  - \( E \ast id + id \)

- A grammar is a right recursive if its production rules can generate a derivation of the form \( A \Rightarrow^* X A \).

- Examples:
  - \( E \rightarrow id \circ E \mid (E) \mid id \)
  - \( E \rightarrow id + F \mid (E) \mid id \)
  - \( F \rightarrow id \ast E \mid id \)
  - \( E \rightarrow id + F \)
  - \( id + id \ast E \)
Parse Tree

- A labeled tree in which
  - the interior nodes are labeled by nonterminals
  - leaf nodes are labeled by terminals
  - the children of an interior node represent a replacement of the associated nonterminal in a derivation
  - corresponding to a derivation

```
E → E + E    (1)
   → id + E    (2)
   → id + E * E (3)
   → id + id * E (4)
   → id + id * id (5)
```

Parse Trees and Derivations

```
E → E + E    (1)
   → id + E    (2)
   → id + E * E (3)
   → id + id * E (4)
   → id + id * id (5)
```

Grammar: Example

List of statements:
- No statement
- One statement: s;
- More than one statement: s; s;
- A statement can be a block of statements. { s; s; } s;

Is the following correct? { s; s; } s; { } s; }
Abstract Syntax Tree
- Representation of actual source tokens
- Interior nodes represent operators.
- Leaf nodes represent operands.

Abstract Syntax Tree for Expression

Abstract Syntax Tree for If Statement
Ambiguous Grammar

- A grammar is ambiguous if it can generate two different parse trees for one string.
- Ambiguous grammars can cause inconsistency in parsing.

Example: Ambiguous Grammar

E → E + E
E → E - E
E → E * E
E → E / E
E → id

Ambiguity in Expressions

- Which operation is to be done first?
  - solved by precedence
    - An operator with higher precedence is done before one with lower precedence.
    - An operator with higher precedence is placed in a rule (logically) further from the start symbol.
  - solved by associativity
    - If an operator is right-associative (or left-associative), an operand in between 2 operators is associated to the operator to the right (left).
    - Right-associated: \( W + (X + (Y + Z)) \)
    - Left-associated: \( ((W + X) + Y) + Z \)
### Precedence

- \( E \rightarrow E + E \)
- \( E \rightarrow E - E \)
- \( E \rightarrow E \ast E \)
- \( E \rightarrow E / E \)
- \( E \rightarrow \text{id} \)
- \( E \rightarrow E + E \)
- \( E \rightarrow E - E \)
- \( E \rightarrow F \)
- \( F \rightarrow F \ast F \)
- \( F \rightarrow F / F \)
- \( F \rightarrow \text{id} \)

### Precedence (cont’d)

- \( E \rightarrow E + E \mid E - E \mid F \)
- \( F \rightarrow F \ast F \mid F / F \mid X \)
- \( X \rightarrow (E) \mid \text{id} \)
- \((\text{id1} + \text{id2}) \ast \text{id3} \ast \text{id4}\)

### Associativity

- **Left-associative operators**
- \( E \rightarrow E + F \mid E - F \mid F \)
- \( F \rightarrow F \ast X \mid F / X \mid X \)
- \( X \rightarrow (E) \mid \text{id} \)
- \((\text{id1} + \text{id2}) \ast \text{id3} / \text{id4}\) = \(((\text{id1} + \text{id2}) \ast \text{id3}) / \text{id4}\)
Ambiguity in Dangling Else

\[ \text{St} \rightarrow \text{IfSt} \mid \ldots \]
\[ \text{IfSt} \rightarrow \text{if ( E ) St} \mid \text{if ( E ) St else St} \]
\[ \text{E} \rightarrow 0 \mid 1 \mid \ldots \]
\[ \text{if ( E ) St} \]
\[ \text{if ( E ) St else St} \]

Disambiguating Rules for Dangling Else

\[ \text{St} \rightarrow \text{MatchedSt} \mid \text{UnmatchedSt} \]
\[ \text{UnmatchedSt} \rightarrow \text{if ( E ) St} \mid \text{if ( E ) MatchedSt else UnmatchedSt} \]
\[ \text{MatchedSt} \rightarrow \text{if ( E ) MatchedSt else MatchedSt} \mid \ldots \]
\[ \text{E} \rightarrow 0 \mid 1 \mid \text{if (0) if (1) St else St} = \text{if (0) if (1) St else St} \]

Extended Backus-Naur Form (EBNF)

- Kleene’s Star/ Kleene’s Closure
  - Seq ::= St {; St}
  - Seq ::= {St ;} St

- Optional Part
  - IfSt ::= if ( E ) St [else St]
  - E ::= F [+ E] | F [- E]
Syntax Diagrams

- Graphical representation of EBNF rules
  - nonterminals: \texttt{\textbf{IfSt}}
  - terminals: \texttt{id}
  - sequences and choices:

- Examples
  - \( X ::= (E) \mid \text{id} \)
  - \( \text{Seq} ::= \{ \text{St} ; \} \text{St} \)
  - \( E ::= F \ [+ E] \)