Lecture 6: Parsing

Administrivia

- We will assign orphans to groups randomly in a few days.
- Josh Hug interviewing today and Tuesday:
  - Vision Seminar: Mon 04:30-05:30 in 380 Soda
  - Undergrads: Tues 09:30-10:15am in 380 Soda
  - Mock Class: Tues 10:30-11:30am in 380 Soda
  - Grads: Tues 03:00-03:45pm in 315 Soda

A Glance at the Map

Source code → Lexical Analysis → Tokens → Parsing → AST → Semantic Analysis → Decorated AST

We are here

Review: BNF

- BNF is another pattern-matching language;
- Alphabet typically set of tokens, such as from lexical analysis, referred to as terminal symbols or terminals.
- Matching rules have form:
  \[ X : \alpha_1\alpha_2 \cdots \alpha_n, \]
  where \( X \) is from a set of nonterminal symbols (or nonterminals or meta-variables), \( n \geq 0 \), and each \( \alpha_i \) is a terminal or nonterminal symbol.
- For emphasis, may write \( X : \epsilon \) when \( n = 0 \).
- Read \( X : \alpha_1\alpha_2 \cdots \alpha_n \) as
  "An \( X \) may be formed from the concatenation of an \( \alpha_1, \alpha_2, \ldots, \alpha_n \)."
- Designate one nonterminal as the start symbol.
- Set of all matching rules is a context-free grammar.

Derivations

- String (of terminals) \( T \) is in the language described by grammar \( G \), \( (T \in L(G)) \) if there is a derivation of \( T \) from the start symbol of \( G \).
- Derivation of \( T = \tau_1 \cdots \tau_k \) from nonterminal \( A \) is sequence of sentential forms:
  \[ A \Rightarrow \alpha_{11}\alpha_{12} \cdots \Rightarrow \alpha_{21}\alpha_{22} \cdots \Rightarrow \cdots \Rightarrow \tau_1 \cdots \tau_k \]
  where each \( \alpha_{ij} \) is a terminal or nonterminal symbol.
- We say that
  \[ \alpha_1 \cdots \alpha_m \beta_1 \cdots \beta_p \Rightarrow \alpha_1 \cdots \alpha_{m-1} \beta_1 \cdots \beta_p \alpha_{m+1} \cdots \alpha_n \]
  if \( B : \beta_1 \cdots \beta_p \) is a production. \( (1 \leq m \leq n) \).
- If \( \Phi \) and \( \Phi' \) are sentential forms, then \( \Phi_1 \Rightarrow \Phi_2 \) means 0 or more \( \Rightarrow \) steps turns \( \Phi_1 \) into \( \Phi_2 \). \( \Phi_1 \Rightarrow \Rightarrow \Phi_2 \) means 1 or more \( \Rightarrow \) steps does it.
- So if \( S \) is start symbol of \( G \), then \( T \in L(G) \) iff \( S \Rightarrow T \).
Example of Derivation

1. $e : s \ ID$
2. $e : s ' ( e ' )$
3. $e : e '/ e$
4. $s : s ( e )$
5. $s : +$
6. $s : -$

Alternative Notation

1. $e : s \ ID$
2. $e : s ( e )$
3. $e : e '/' e$
4. $s : \epsilon | + | -$

Problem: Derive $- ID / ( ID / ID )$

$$e \Rightarrow e / e \Rightarrow s \ ID / e \Rightarrow ID / s ( e )$$
$$\Rightarrow ID / ( e ) \Rightarrow ID / ( e / e ) \Rightarrow ID / ( s \ ID / e )$$
$$\Rightarrow ID / ( ID / e ) \Rightarrow ID / ( ID / s \ ID )$$
$$\Rightarrow ID / ( ID / ID )$$

Types of Derivation

- **Context free** means can replace nonterminals in any order (i.e., regardless of context) to get same result (as long as you use same productions).
- So, if we use a particular rule for selecting nonterminal to "produce" from, can characterize derivation by just listing productions.
- Previous example was leftmost derivation: always choose leftmost nonterminals. Completely characterized by list of productions: 3, 1, 6, 2, 4, 3, 1, 4, 1, 4.

Derivations and Parse Trees

- A leftmost derivation also completely characterized by parse tree:

```
 e
 / \
 e / e / e / s ( e ) / s ( e )
 / \
 s ( e ) / s ( e ) / s ( e )
 / \
 ID / ID / ID / ID
```

What is the rightmost derivation for this?

$$e \Rightarrow e / e \Rightarrow e / s ( e ) \Rightarrow e / s ( e / e )$$
$$\Rightarrow e / s ( e / e ) \Rightarrow e / s ( e / e )$$
$$\Rightarrow e / s ( e / e ) \Rightarrow e / s ( e / e )$$
$$\Rightarrow e / s ( e / e ) \Rightarrow e / s ( e / e )$$

Ambiguity

- Only one derivation for previous example.
- What about $ID / ID / ID$?
- Claim there are two parse trees, corresponding to two leftmost derivations. What are they?

- If there exists even one string like $ID / ID / ID$ in $L(G)$, we say $G$ is ambiguous (even if other strings only have one parse tree).
Ambiguity

- Only one derivation for previous example.
- What about ‘ID / ID / ID’?
- Claim there are two parse trees, corresponding to two leftmost derivations. What are they?

- If there exists even one string like ID / ID / ID in $L(G)$, which is ambiguous (even if other strings only have one parse tree).
Review: Syntax-Directed Translation

- Want the structure of sentences, not just whether they are in the language, because this drives translation.
- Associate translation rules to each production, just as Flex associated actions with matching patterns.
- Bison notation:

  ```
  e : e '/' e { $$ = doDivide($1, $3); }
  ```

  provides way to refer to and set semantic values on each node of a parse tree.
- Compute these semantic values from leaves up the parse tree.
- Same as the order of a rightmost derivation in reverse (a.k.a a canonical derivation).
- Alternatively, just perform arbitrary actions in the same order.

Example: Conditional statement

**Problem:** if-else or if-elif-else statements in Python (else optional).
Assume that only (indented) suites may be used for then and else clauses, that nonterminal `stmt` defines an individual statement (one per line), and that nonterminal `expr` defines an expression. Lexer supplies INDENTS and DEDENTS. A `cond` is a kind of `stmt`.

```c
expr : ...
stmt : ... | cond | ...
cond : "if" expr ':' suite elifs elsestmts: stmt | stmts stmt
eifs: ε | "elif" expr ':' suite elifs
eelse : ε | "else" ':' suite
```

But this doesn't quite work: recognizes correct statements and rejects incorrect ones, but is ambiguous. E.g.,

```c
if (foo) if (bar) walk(); else chewGum();
```

Do we chew gum if foo is false? That is, is this equivalent to

```c
if (foo) { if (bar) walk(); } else chewGum(); /*or*/ if (foo) { if (bar) walk(); else chewGum(); } ?
```

Example resolved: Conditional statement in Java

The rule is supposed to be "each 'else' attaches to the nearest open 'if' on the left," which is captured by:

```c
expr : ...
stmt : ... | cond | ...
cond : "if" '(' expr ')' stmt closedelseelse : ε | "else" stmt
closed : "if" '(' expr ')' stmt closed "else" stmt closedcond : "if" '(' expr ')' stmt
| "if" '(' expr ')' stmt closed "else" stmt
closed
```

This does not allow us to interpret

```c
if (foo) if (bar) walk(); else chewGum();
as
if (foo) { if (bar) walk(); } else chewGum();
```

But it's not exactly clear, is it?
Puzzle: NFA to BNF

Problem: What BNF grammar accepts the same string as this NFA?

A conventional answer (from class):

\[ S : S2S \mid S3S S2 : 1 \mid 11 S3 : 1 \mid 111 \]

General answer (adaptable to any NFA), with one nonterminal per state:

\[ S0 : S1 \mid S4 S4 : 1 \mid 0 \]
\[ S1 : 0 \mid S2 \mid 0 \mid S1 \]
\[ S2 : 1 \mid S3 \mid S2 S2 \]
\[ S3 : S1 \mid 0 \mid S3 S3 \]
\[ S7 : S4 \mid 0 \mid S7 \mid \epsilon \]

Nonterminal \( S_k \) is "the set of strings that will get me from \( S_k \) in the NFA to a final state in the NFA."