

CSE 582 – Compilers

Parsing & Context-Free Grammars
Hal Perkins
Autumn 2002

Agenda for Today

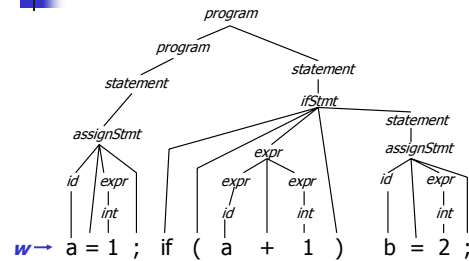
- n Parsing overview
- n Context free grammars
- n Ambiguous grammars

Parsing

- n The syntax of most programming languages can be specified by a *context-free grammar* (CGF)
- n Parsing: Given a grammar G and a sentence w in $L(G)$, traverse the derivation (parse tree) for w in some *standard order* and do *something useful* at each node
 - n The tree might not be produced explicitly, but the control flow of a parser corresponds to a traversal

Old Example

```
program ::= statement | program statement
statement ::= assignStmt | ifStmt
assignStmt ::= id = expr ;
ifStmt ::= if ( expr ) stmt
expr ::= id | int | expr + expr
id ::= a | b | c | i | j | k | n | x | y | z
int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```



“Standard Order”

- n For practical reasons we want the parser to be *deterministic* (no backtracking), and we want to examine the source program from *left to right*.
 - n (i.e., parse the program in linear time in the order it appears in the source file)

Common Orderings

- n Top-down
 - n Start with the root
 - n Traverse the parse tree depth-first, left-to-right (leftmost derivation)
 - n LL(k)
- n Bottom-up
 - n Start at leaves and build up to the root
 - n Effectively a rightmost derivation in reverse(!)
 - n LR(k) and subsets (LALR(k), SLR(k), etc.)

"Something Useful"

- At each point (node) in the traversal, perform some *semantic action*
 - Construct nodes of full parse tree (rare)
 - Construct abstract syntax tree (common)
 - Construct linear, lower-level representation (more common in later parts of a modern compiler)
 - Generate target code on the fly (1-pass compiler; not common in production compilers – can't generate very good code in one pass)

10/3/2002

© 2002 Hal Perkins & UW CSE

C-7

Context-Free Grammars

- Formally, a grammar G is a tuple $\langle N, \Sigma, P, S \rangle$ where
 - N a finite set of non-terminal symbols
 - Σ a finite set of terminal symbols
 - P a finite set of productions
 - A subset of $N \times (N \cup \Sigma)^*$
 - S the *start symbol*, a distinguished element of N
 - If not specified otherwise, this is usually assumed to be the non-terminal on the left of the first production

10/3/2002

© 2002 Hal Perkins & UW CSE

C-8

Standard Notations

- a, b, c elements of Σ
- w, x, y, z elements of Σ^*
- A, B, C elements of N
- X, Y, Z elements of $N \cup \Sigma$
- α, β, γ elements of $(N \cup \Sigma)^*$
- $A \rightarrow \alpha$ or $A ::= \alpha$ if $\langle A, \alpha \rangle$ in P

10/3/2002

© 2002 Hal Perkins & UW CSE

C-9

Derivation Relations (1)

- $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ iff $A ::= \beta$ in P
 - derives
- $A \Rightarrow^* w$ if there is a chain of productions starting with A that generates w
 - transitive closure

10/3/2002

© 2002 Hal Perkins & UW CSE

C-10

Derivation Relations (2)

- $w A \gamma \Rightarrow_{lm} w \beta \gamma$ iff $A ::= \beta$ in P
 - derives leftmost
- $\alpha A w \Rightarrow_{rm} \alpha \beta w$ iff $A ::= \beta$ in P
 - derives rightmost
- We will only be interested in leftmost and rightmost derivations – not random orderings

10/3/2002

© 2002 Hal Perkins & UW CSE

C-11

Languages

- For A in N , $L(A) = \{ w \mid A \Rightarrow^* w \}$
- If S is the start symbol of grammar G , define $L(G) = L(S)$

10/3/2002

© 2002 Hal Perkins & UW CSE

C-12

Reduced Grammars

- Grammar G is *reduced* iff for every production $A ::= \alpha$ in G there is a derivation
 - $S \Rightarrow^* x A z \Rightarrow^* x \alpha z \Rightarrow^* xyz$
 - i.e., no production is useless
- Convention: we will use only reduced grammars

10/3/2002 © 2002 Hal Perkins & UW CSE C-13

Ambiguity

- Grammar G is *unambiguous* iff every w in $L(G)$ has a unique leftmost (or rightmost) derivation
 - Fact: unique leftmost or unique rightmost implies the other
- A grammar without this property is *ambiguous*
 - Note that other grammars that generate the same language may be unambiguous
- We need unambiguous grammars for parsing

10/3/2002 © 2002 Hal Perkins & UW CSE C-14

Example: Ambiguous Grammar for Arithmetic Expressions

$$\begin{aligned}
 \text{expr} &::= \text{expr} + \text{expr} \mid \text{expr} - \text{expr} \\
 &\quad \mid \text{expr} * \text{expr} \mid \text{expr} / \text{expr} \mid \text{int} \\
 \text{int} &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
 \end{aligned}$$

- Exercise: show that this is ambiguous
 - How? Show two different leftmost or rightmost derivations for the same string
 - Equivalently: show two different parse trees for the same string

10/3/2002 © 2002 Hal Perkins & UW CSE C-15

Example (cont)

- Give a leftmost derivation of $2+3*4$ and show the parse tree

10/3/2002 © 2002 Hal Perkins & UW CSE C-16

Example (cont)


- Give a different leftmost derivation of $2+3*4$ and show the parse tree

10/3/2002 © 2002 Hal Perkins & UW CSE C-17

Another example

- Give two different derivations of $5+6+7$


10/3/2002 © 2002 Hal Perkins & UW CSE C-18



What's going on here?

- The grammar has no notion of precedence or associativity
- Solution
 - Create a non-terminal for each level of precedence
 - Isolate the corresponding part of the grammar
 - Force the parser to recognize higher precedence subexpressions first

10/3/2002 © 2002 Hal Perkins & UW CSE C-19




Classic Expression Grammar

```


expr ::= expr + term | expr - term | term
term ::= term * factor | term / factor | factor
factor ::= int | ( expr )
int ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
  
```

10/3/2002 © 2002 Hal Perkins & UW CSE C-20



Check: Derive $2 + 3 * 4$


10/3/2002 © 2002 Hal Perkins & UW CSE C-21



Check: Derive $5 + 6 + 7$


- Note interaction between left- vs right-recursive rules and resulting associativity

10/3/2002 © 2002 Hal Perkins & UW CSE C-22



Check: Derive $5 + (6 + 7)$

10/3/2002 © 2002 Hal Perkins & UW CSE C-23



Another Classic Example

- Grammar for conditional statements


```

ifStm ::= if ( cond ) stm
          | if ( cond ) stm else stm
      
```
- Exercise: show that this is ambiguous
 - How?

10/3/2002 © 2002 Hal Perkins & UW CSE C-24

ifStm ::= if (*cond*) *stm*
 | if (*cond*) *stm*
 else *stm*

One Derivation

if (*cond*) if (*cond*) *stm* else *stm*

10/3/2002 © 2002 Hal Perkins & UW CSE C-25

ifStm ::= if (*cond*) *stm*
 | if (*cond*) *stm*
 else *stm*

Another Derivation

if (*cond*) if (*cond*) *stm* else *stm*

10/3/2002 © 2002 Hal Perkins & UW CSE C-26

Solving if Ambiguity

- Fix the grammar to separate if statements with else clause and if statements with no else
 - Done in Java reference grammar
 - Adds lots of non-terminals
- Use some ad-hoc rule in parser
 - "else matches closest unpaired if"

10/3/2002 © 2002 Hal Perkins & UW CSE C-27

Parser Tools and Operators

- Most parser tools can cope with ambiguous grammars
 - Makes life simpler if used with discipline
- Typically one can specify operator precedence & associativity

10/3/2002 © 2002 Hal Perkins & UW CSE C-28

Parser Tools and Ambiguous Grammars

- Possible rules for resolving other problems
 - Earlier productions in the grammar preferred to later ones
 - Longest match used if there is a choice
- The tools we will use allow for this
 - But be sure that what the tool does is really what you want

10/3/2002 © 2002 Hal Perkins & UW CSE C-29

Coming Attractions

- Next topic: LR parsing
 - Continue reading ch. 3

10/3/2002 © 2002 Hal Perkins & UW CSE C-30