



CSE 413 Spring 2011

Parsing & Context-Free Grammars



Agenda

- Parsing overview
- Context free grammars
- Ambiguous grammars

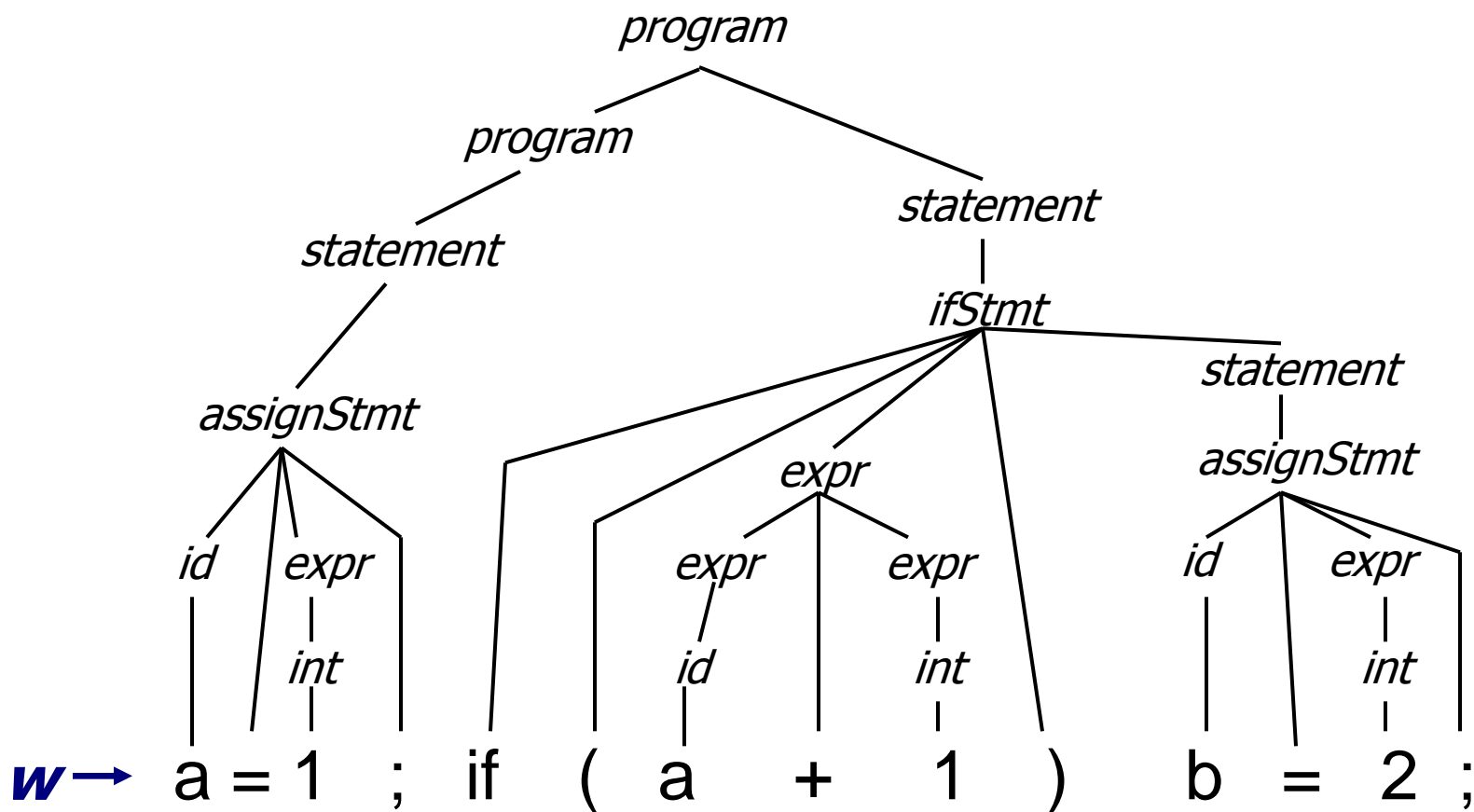
Parsing

- The syntax of most programming languages can be specified by a *context-free grammar* (CGF)
- **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
 - The tree might not be produced explicitly, but the control flow of a parser corresponds to a traversal

Old Example

G

```
program ::= statement | program statement
statement ::= assignStmt | ifStmt
assignStmt ::= id = expr ;
ifStmt ::= if ( expr ) statement
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”

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

Common Orderings

■ Top-down

- Start with the root
- Traverse the parse tree depth-first, left-to-right (leftmost derivation)
- LL(k)

■ Bottom-up

- Start at leaves and build up to the root
 - Effectively a rightmost derivation in reverse(!)
- 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 or interpret on the fly (1-pass compiler & interpreters; not common in production compilers – but what we will do for our project)

Context-Free Grammars (review)

- 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

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

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

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
- Parsers normally work with only leftmost or rightmost derivations – not random orderings

Languages

- For A in N , $L(A) = \{ w \mid A \Rightarrow^* w \}$
 - i.e., set of strings (words, terminal symbols) generated by nonterminal A
- If S is the start symbol of grammar G , we define $L(G) = L(S)$

Reduced Grammars

- Grammar G is *reduced* iff for every production $A ::= \alpha$ in G there is some 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

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

Example

- Top down, Leftmost derivation for: **a = 1 + b ;**

Example

- Grammar

$S ::= aABe$

$A ::= Abc \mid b$

$B ::= d$

- Top down, leftmost derivation of: **abbcde**

Ambiguity

- Grammar G is *unambiguous* iff every w in $L(G)$ has a unique leftmost (or rightmost) derivation
 - Fact: either 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

Example: Ambiguous Grammar for Arithmetic Expressions

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

- 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

Example (cont)

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

Example (cont)

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

Another example

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

What's going on here?

- This grammar has no notion of precedence or associativity
- Standard 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

Classic Expression Grammar

$expr ::= expr + term \mid expr - term \mid term$

$term ::= term * factor \mid term / factor \mid factor$

$factor ::= int \mid (expr)$

$int ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7$



Check: Derive $2 + 3 * 4$



Check: Derive $5 + 6 + 7$

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



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

Another Classic Example

- Grammar for conditional statements

$stmt ::= \text{if } (cond) stmt$

 | $\text{if } (cond) stmt \text{ else } stmt$

 | $assign$

- Exercise: show that this is ambiguous

- How?

$stmt ::=$ if (*cond*) *stmt*
| if (*cond*) *stmt* else *stmt*
| *assign*

One Derivation

if (*cond*) if (*cond*) *stmt* else *stmt*

$stmt ::= if (cond) stmt$
 $| if (cond) stmt \text{ else } stmt$
 $| assign$

Another Derivation

$if (cond) if (cond) stmt \text{ else } stmt$

Solving if Ambiguity

- Fix the grammar to separate if statements with else from if statements with no else
 - Done in original Java reference grammar
 - Adds lots of non-terminals
 - Need productions for things like “while statement that has unmatched if” and “while statement with only matched ifs”, etc. etc. etc.
- Use some ad-hoc rule in parser
 - “else matches closest unpaired if”

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
 - Allows simpler, ambiguous grammar with fewer nonterminals as basis for generated parser, without creating problems

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
- Parser tools normally allow for this
 - But be sure that what the tool does is really what you want



Or...

- If the parser is hand-written, either fudge the grammar or the parser or cheat where it helps.

to be continued...