CSE 3302
Programming Languages

Syntax (cont.)

Chengkai Li, Weimin He
Spring 2008

What is Parsing?

- **Given a grammar and a token string:**
  - determine if the grammar can generate the token string?
  - i.e., is the string a legal program in the language?
- **In other words, to construct a parse tree for the token string.**

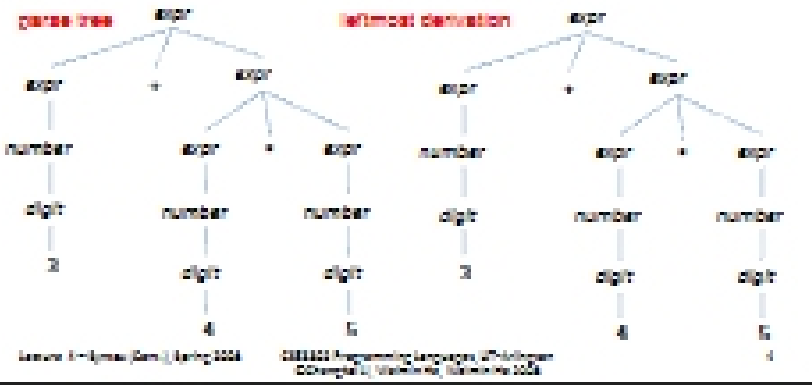
What's significant about parse tree?

A parse tree gives a unique syntactic structure

- Leftmost, rightmost derivation
- There is only one leftmost derivation for a parse tree, and symmetrically only one rightmost derivation for a parse tree.

Example

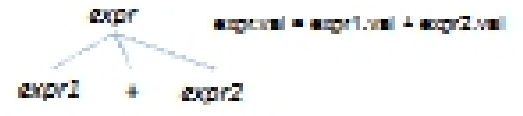
expr → *expr* + *expr* | *expr* * *expr* | (*expr*) | *number*
number → *number digit* | *digit*
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



What's significant about parse tree?

A parse tree has a unique meaning, thus provides basis for semantic analysis.

(Syntax-directed semantics: semantics are attached to syntactic structure.)



Relationship among language, grammar, parser

- Chomsky Hierarchy
  http://en.wikipedia.org/wiki/Chomsky_hierarchy
- A language can be described by multiple grammars, while a grammar defines one language.
- A grammar can be parsed by multiple parsers, while a parser accepts one grammar, thus one language.
- Should design a language that allows simple grammar and efficient parser
- For a language, we should construct a grammar that allows fast parsing
- Given a grammar, we should build an efficient parser

Ambiguity

- **Ambiguous grammar**: There can be multiple parse trees for the same sentence (program)
  - In other words, multiple leftmost derivations.
- Why is it bad?
  - Multiple meanings

Ambiguity

- **Was this ambiguous?**

number → *number digit* | *digit*

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- **How about this?**

expr → *expr* – *expr* | *number*

Deal with Ambiguity

- **Unambiguous Grammar**
  - Rewrite the grammar to avoid ambiguity.

Example of Ambiguity: Precedence

expr → *expr* + *expr* | *expr* * *expr* | (*expr*) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Two different parse trees for expression 3+4*5

Eliminating Ambiguity for Precedence

- Establish "precedence cascade": using different structured names for different constructs, adding grammar rules.
  - Higher precedence : lower in cascade

expr → *expr* + *expr* | *expr* * *expr* | (*expr*) | *number*

⇒

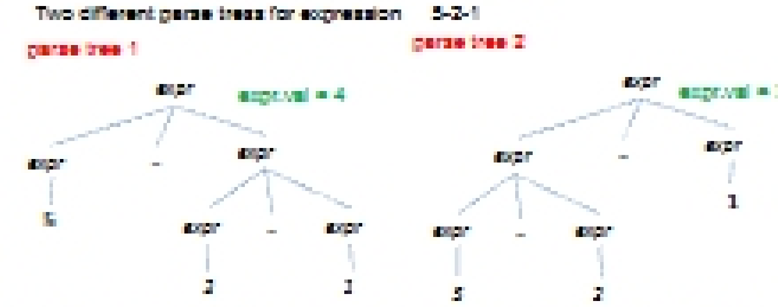
expr → *expr* + *expr* | *term*

term → *term* * *term* | (*expr*) | *number*

Example of Ambiguity: Associativity

expr → *expr* – *expr* | (*expr*) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Two different parse trees for expression 5-2-1



– is right-associative, which is against common practice in integer arithmetic

– is left-associative, which is what we usually assume

Associativity

- Left-Associative: + - * /
- Right-Associative: =

  What is meant by a=b=c=1?

Eliminating Ambiguity for Associativity

- **left-associativity: left-recursion**

expr → *expr* – *expr* | (*expr*) | *number*

→

expr → *expr* – *term* | *term*

term → (*expr*) | *number*

- **right-associativity: right-recursion**

expr → *expr* = *expr* | a | b | c

→

expr → *term* = *expr* | *term*

term → a | b | c

Putting Together

expr → *expr* – *expr* | *expr* / *expr* | (*expr*) | *number*

number → *number digit* | *digit*

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

We want to make - left-associative and / has precedence over -

→

expr → *expr* – *term* | *term*

term → *term* / *factor* | *factor*

factor → (*expr*) | *number*

number → *number digit* | *digit*

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Example of Ambiguity: Dangling-Else

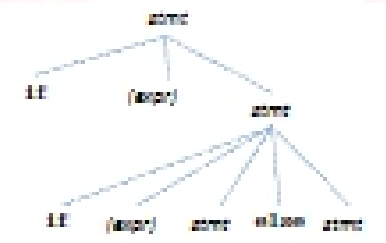
stmt → if (*expr*) *stmt*

| if (*expr*) *stmt* else *stmt*

| *other-stmt*

Two different parse trees for "if(expr) if(expr) stmt else stmt"

parse tree 1



parse tree 2

Eliminating Dangling-Else

stmt → *matched_stmt*

| *unmatched_stmt*

matched_stmt → if (*expr*) *matched_stmt* else *matched_stmt*

| *other-stmt*

unmatched_stmt → if (*expr*) *stmt*

| if (*expr*) *matched_stmt* else *unmatched_stmt*

EBNF

- Repetition { }

number → *digit* | *number digit* → *number* → *digit*{}

expr → *expr* – *term* | *term* → *expr* → *term* {– *term*}

- Option [ ]

signed-number → *sign number* | *number* → *signed-number* → [*sign*] *number*

if-stmt → if (*expr*) *stmt* → *if-stmt* → if (*expr*) *stmt* [else *stmt*]

| if (*expr*) *stmt* else *stmt*