


CSE 3302 Programming Languages




Syntax (cont.)

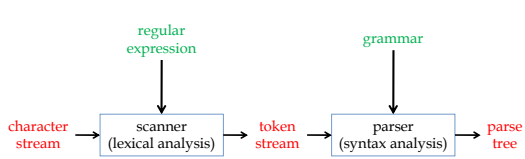
Chengkai Li
Fall 2007

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 1

Review of Lecture 3




- Scanning and Parsing
 - Lexical Structure: The structure of tokens (words)
 - Syntactical Structure: The structure of programs



Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 2

Automatic Tools



regular expression description of the tokens
→ (lex/flex)
scanner of a language


- Example: Figure 4.1 (page 82)

Context-Free Grammar
→ (Yacc/Bison)
parser

- Example: Figure 4.13 (page 112)

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 3


Review of Lecture 3



- Tokens:**
 - Reserved words (keywords)
 - Literals/constants
 - Special symbols
 - Identifiers
- Principle of Longest Substring**
 - The longest possible string of characters is collected into a single token.
 - The principle requires that tokens are separated by white spaces.

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 4

Review of Lecture 3




- Context-Free Grammar** (Backus-Naur Forms (BNF))
 - Grammar rules** (productions):
 - left-hand side: single nonterminal
 - nonterminals** (structured names)
 - terminals** (tokens)
 - start symbol**
- Language:** the set of token strings that can be produced by **derivation** from the start symbol
- Derivation:**

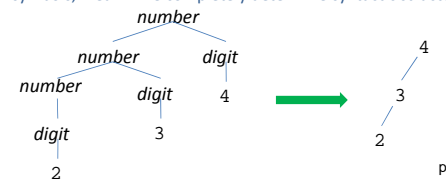
$number \Rightarrow number\ digit \Rightarrow number\ digit\ digit \Rightarrow digit\ digit\ digit \Rightarrow 2\ digit\ digit \Rightarrow 23\ digit \Rightarrow 234$

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 5

Review of Lecture 3



- Derivations:** different derivations for the same syntactic structure.
- Parse trees:** capture intrinsic structure, more intuitive, still tedious.
- Abstract Syntax Trees (Syntax Trees):** Remove "unnecessary" symbols, meanwhile completely determine syntactic structure.



page 91

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 6

Topics Today



- Topics:
 - Ambiguities in Grammar
 - Parsing Techniques
- Intuitive analysis and conclusion
- No formal theorems and rigorous proofs
- More details: compilers, automata theory

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?



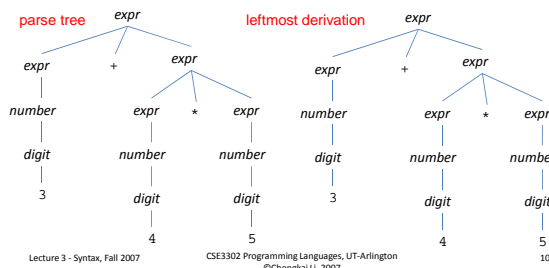
A parse tree gives a unique syntactic structure

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

Example



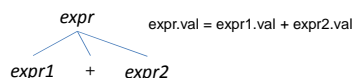
$expr \rightarrow expr + expr \mid expr * expr \mid (expr) \mid number$
 $number \rightarrow number digit \mid digit$
 $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$



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 it is bad?**
 - Multiple meaning
- Multiple derivation can be ok, **but multiple leftmost derivation is the same as multiple parse tree.**

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 13

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
```

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 14

Deal with Ambiguity

- **disambiguating rules:**
 - Use semantics in determining which parse tree to construct

OR

- **unambiguous grammar**
 - Rewrite the grammar to avoid ambiguity.

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 15

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

parse tree 1

parse tree 2

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 16

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
```

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 17

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

parse tree 1

expr.val = 4

parse tree 2

expr.val = 2

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

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

Lecture 3 - Syntax, Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2007 18

Associativity



- Left-Associative: + - * /
 - Right-Associative: =
- What is meant by $a=b=c=1$?

Lecture 3 - Syntax, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

19

Eliminating Ambiguity for Associativity



- left-associativity: left-recursion
- $$expr \rightarrow expr - expr \mid (expr) \mid number$$
-
- $$expr \rightarrow expr - term \mid term$$
- $$term \rightarrow (expr) \mid number$$
- right-associativity: right-recursion
- $$expr \rightarrow expr = expr \mid a \mid b \mid c$$
-
- ?

Lecture 3 - Syntax, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

20

Putting Together



$$expr \rightarrow expr + expr \mid expr * expr \mid (expr) \mid number$$

$$number \rightarrow number digit \mid digit$$

$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

We want to make + left-associative, and * has higher precedence than +



?

Lecture 3 - Syntax, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

21

Example of Ambiguity: Dangling-Else

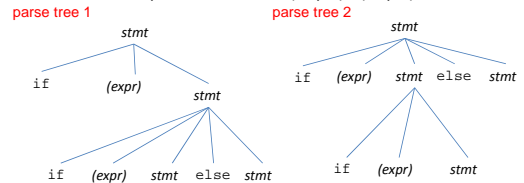


$$stmt \rightarrow \text{if} (expr) stmt$$

$$\mid \text{if} (expr) stmt \text{ else } stmt$$

$$\mid \text{other-stmt}$$

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



Lecture 3 - Syntax, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

22

Eliminating Dangling-Else



$$stmt \rightarrow \text{matched_stmt}$$

$$\mid \text{unmatched_stmt}$$

$$\text{matched_stmt} \rightarrow \text{if} (expr) \text{matched_stmt} \text{ else } \text{matched_stmt}$$

$$\mid \text{other-stmt}$$

$$\text{unmatched_stmt} \rightarrow \text{if} (expr) stmt$$

$$\mid \text{if} (expr) \text{matched_stmt} \text{ else } \text{unmatched_stmt}$$

Lecture 3 - Syntax, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

23