CS412/413

Introduction to Compilers

Radu Rugina

Lecture 7: LL Parsing Tables

6 Feb 2006

Outline

- Building LL parsing tables
- FIRST/FOLLOW sets
- Making grammars LL(1)

Implementing A Top-Down Parser

- LL(1) grammar example:

  S → ES'
  S' → ε | + S
  E → num | (S)

- Use a predictive parsing table:

	num	+	(	)	$
S	→ ES'		→ ES'		
S'		→ +S		→ ε	→ ε
E	→ **num**		→ (S)		

- Implement a recursive-descent parser using mutually recursive procedures `parse_S()`, `parse_S'()`, `parse_E()`

How to Construct Parsing Tables

- Need an algorithm that generates a predictive parse table from a grammar

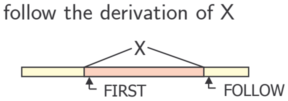
S → ES'
S' → ε | + S
E → number | (S)

?

	num	+	(	)	$
S	ES'		ES'		
S'		+S		ε	ε
E	num		(S)		

Constructing Parse Tables

- Parsing table tells us, for each non-terminal and each look-ahead symbol what production to use
- FIRST(γ) for arbitrary string of terminals and non-terminals γ = set of symbols that might begin the fully expanded version of γ
- FOLLOW(X) for a non-terminal X = set of symbols that might follow the derivation of X



Parse Table Entries

- Consider a production X → γ
- Add → γ to the X row for symbols in FIRST(γ)

	num	**+**	**(**	**)**	**$**
S	→ ES'		→ ES'		
S'		→ +S		→ ε	→ ε
E	→ num		→ (S)		

- If γ can derive ε (γ is nullable), add → γ for each symbol in FOLLOW(X)
- Grammar is LL(1) if no conflicting entries

Computing nullable, FIRST

- X is nullable if it can derive the empty string:
  - if : $X \to \varepsilon$
  - if : $X \to Y_1 \dots Y_n$ where all Y_i are nullable
  - Algorithm: assume all non-terminals non-nullable, apply rules repeatedly until no change
- Computing FIRST(γ)
  - FIRST(X) $\supseteq$ FIRST(γ)   if $X \to \gamma$
  - FIRST(**a** β) = { **a** }
  - FIRST(X β) $\supseteq$ FIRST(X)
  - FIRST(X β) $\supseteq$ FIRST(β) if X is nullable
  - Algorithm: Assume FIRST(γ) = {} for all γ, apply rules repeatedly to build FIRST sets.

Computing FOLLOW

- Compute FOLLOW(X):
  - FOLLOW(S) $\supseteq$ { \$ }
  - If $X \to \alpha Y \beta$, FOLLOW(Y) $\supseteq$ FIRST(β)
  - If $X \to \alpha Y \beta$ and β is nullable (or non-existent), FOLLOW(Y) $\supseteq$ FOLLOW(X)
- Algorithm: Assume FOLLOW(X) = { } for all X, apply rules repeatedly to build FOLLOW sets
- Common theme: iterative analysis. Start with initial assignment, apply rules until no change

Example

S → E S ′
S ′ → ε | **+** S
E → **num** | **(** S **)**

- nullable
  - only S ′ is nullable
- FIRST
  - FIRST(E S') = {**num**, **(** }
  - FIRST(+S) = { **+** }
  - FIRST(**num**) = {**num**}
  - FIRST(**(**S**)**) = { **(** },
  - FIRST(S ′) = { + }
- FOLLOW
  - FOLLOW(S) = { **\$**, **)** }
  - FOLLOW(S′) = {**\$**, **)**}
  - FOLLOW(E) = { **+**, **)**, **\$**}

	num	**+**	**(**	**)**	**\$**
S	→ E S′		→ E S′		
S′		→ **+**S		→ ε	→ ε
E	→ num		→ (S)		

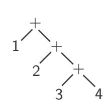
LL Grammars and Associativity

- We have been using grammar for language of "sums with parentheses" e.g., (1+(3+4))+5
- Started with simple, right-associative grammar:
  S → E + S | E
  E → num | (S)
- Transformed it to an LL(1) grammar by left-factoring:
  S → ES'
  S' → ε | + S
  E → num | (S)
- What if we start with a left-associative grammar?
  S → S + E | E
  E → num | (S)

Left vs. Right Associativity

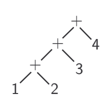
Right recursion : right-associative

S → E + S
S → E
E → num



Left recursion : left-associative

S → S + E
S → E
E → num



Left Recursion

- Left-recursive grammars don't work with top-down parsing: we don't know where to stop the recursion

derived string	lookahead	read/not read
S	1	1 + 2 + 3 + 4
S + E	1	1 + 2 + 3 + 4
S + E + E	1	1 + 2 + 3 + 4
S + E + E + E	1	1 + 2 + 3 + 4
E + E + E + E	1	1 + 2 + 3 + 4
1 + E + E + E	2	1 + 2 + 3 + 4
1 + 2 + E + E	3	1 + 2 + 3 + 4
1 + 2 + 3 + E	4	1 + 2 + 3 + 4
1 + 2 + 3 + 4	\$	1 + 2 + 3 + 4

Left-Recursive Grammars

- Left-recursive grammars are not LL(1) !

  $S \rightarrow S\ \alpha$
  $S \rightarrow \beta$

- FIRST(β) $\subseteq$ FIRST($S\alpha$)
- If β is nullable, then so is $S\alpha$
- Both productions will appear in the table at row S in all the columns corresponding to symbols in FIRST(β) if β is not nullable, or to symbols in FOLLOW(S) if β is nullable

Eliminate Left Recursion

- Method for left-recursion elimination:

  Replace

  $A \rightarrow A\ \alpha_1 \mid \ldots \mid A\ \alpha_m$
  $A \rightarrow \beta_1 \mid \ldots \mid \beta_n$

  with

  $A \rightarrow \beta_1\ B \mid \ldots \mid \beta_n\ B$
  $B \rightarrow \alpha_1\ B \mid \ldots \mid \alpha_m\ B \mid \varepsilon$

- (See the complete algorithm in the Dragon Book)

Creating an LL(1) Grammar

- Start with a left-recursive grammar:

  $S \rightarrow S + E$
  $S \rightarrow E$

  and apply left-recursion elimination algorithm:

  $S \rightarrow E S'$
  $S' \rightarrow +E\ S' \mid \varepsilon$

- Start with a right-recursive grammar:

  $S \rightarrow E + S$
  $S \rightarrow E$

  and apply left-factoring to eliminate common prefixes:

  $S \rightarrow E\ S'$
  $S' \rightarrow +\ S \mid \varepsilon$

Top-Down Parsing Summary