CS 241 – Week 8 Tutorial Solutions

Top-Down Parsing: LL(1) Parsing

Spring 2015

Solutions

1. To implement the predictor table for the grammar, we first need to generate *nullable*, $First$, $Follow$ for all production rules, non-terminals and terminals in the grammar.

   For example,

$$\begin{aligned} First(pX) &= \{p\} \\ First(q) &= \{q\} \\ First(XY) &= \{p, q\} \\ First(aXYb) &= \{a\} \\ First(\vdash S \dashv) &= \{\vdash\} \end{aligned}$$

   and

$$\begin{aligned} Follow(S') &= \{\} \\ Follow(S) &= \{\dashv\} \\ Follow(X) &= \{q, b, \dashv\} \\ Follow(Y) &= \{b, \dashv\} \end{aligned}$$

   The completed table:

	$\vdash$	$\dashv$	a	b	p	q
S'	$S' \to \vdash S \dashv$					
S		$S \to XY$	$S \to aXYb$		$S \to XY$	$S \to XY$
X		$X \to \epsilon$		$X \to \epsilon$	$X \to pX$	$X \to \epsilon$
Y		$Y \to \epsilon$		$Y \to \epsilon$		$Y \to q$

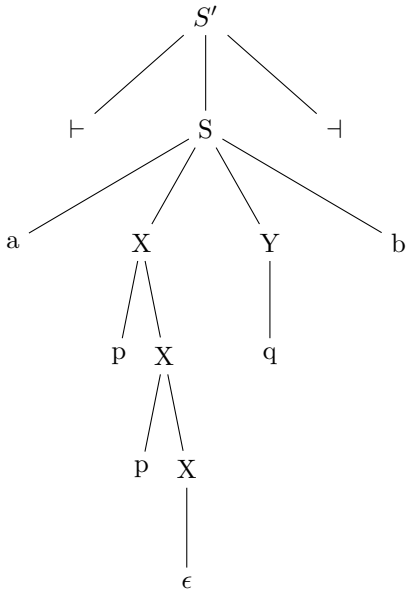
2. **Algorithm:**

   If there is a terminal on top of the stack, we consume it. If there is a nonterminal, we look in the predictor table (using the terminal at the front of the remaining input) to find the rule to expand it with. We repeat until we have consumed the whole input string. We accept when $\dashv$ is read. If at any point there is no rule to expand with, then the input string does not belong to the language and we reject the input string.

   Trace of the algorithm:

Action	Consumed Input	Stack	Remaining Input
initialize		S'	⊢appqb⊣
S → ⊢S⊣		⊢S⊣	⊢appqb⊣
read ⊢	⊢	S⊣	appqb⊣
S → aXYb	⊢	aXYb⊣	appqb⊣
read a	⊢a	XYb⊣	ppqb⊣
X → pX	⊢a	pXYb⊣	ppqb⊣
read p	⊢ap	XYb⊣	pqb⊣
X → pX	⊢ap	pXYb⊣	pqb⊣
read p	⊢app	XYb⊣	qb⊣
X →	⊢app	Yb⊣	qb⊣
Y → q	⊢app	qb⊣	qb⊣
read q	⊢appq	b⊣	b⊣
read b	⊢appqb	⊣	⊣
read ⊣	⊢appqb⊣		

Reading the production rules that were applied from top to bottom gives a forward left-canonical derivation. We can use this derivation to obtain the parse tree.



3. This question is a bit tricky. Suppose you are trying to parse a word in this language with an LL(1) parser. You currently have 'S' on top of your stack, and all you know about the input is that the first character is 'p' (since you only have one symbol of lookahead, you can't make decisions based on the full input string).

   If you see a 'p', the obvious thing to do is apply the rule $S \to XY$ then the rule $X \to pX$ then consume the p from the input.

Action	Stack	First symbol of remaining input
	S	p
S → XY	XY	p
X → pX	pXY	p
read p	XY	a

   Now the next character in the input is 'a'. What do you do? There is no way to generate an 'a' from

the nonterminals X and Y.

In this particular case, the full input word was "pab". If you knew this, you could have first used the rule S $\vdash$ Sab and parsed it properly. But with only one symbol of lookahead, there is no way to tell which rule to expand S by.

In general, if you have multiple production rules from the same non-terminal, and one of those is left-recursive, your grammar will not be LL(1). Example:

expr $\rightarrow$ expr + term
expr $\rightarrow$ term
term $\rightarrow$ ID