Assignment 2: Parsing

15-411: Compiler Design
Nathan Snyder (npsnyder@andrew) and Anand Subramanian(asubrama@andrew)

Due: Tuesday, September 21, 2010 (1:30 pm)

Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own. You may hand in a handwritten solution or a printout of a typeset solution at the beginning of lecture on Tuesday, September 21. Please read the late policy for written assignments on the course web page. If you decide not to typeset your answers, make sure the text and pictures are legible and clear.

Overview

This assignment is designed to test your understanding of the concepts that underly LL and LR parsing. In particular, it is here to ensure that you understand the details that widely available tools such as parser generators abstract away from you when you work on the Labs. Being well-versed in the concepts behind LL and LR parsing can often be crucial for using parser generators effectively.

In this assignment, you will be working with the context free grammar Λ, defined as follows:

$$
\begin{array}{rcll}
\gamma_1 & : & \langle \text{E} \rangle & \rightarrow \quad \text{x} \\
\gamma_2 & : & \langle \text{E} \rangle & \rightarrow \quad \lambda\ \text{x} : \langle \text{E} \rangle \\
\gamma_3 & : & \langle \text{E} \rangle & \rightarrow \quad \langle \text{E} \rangle\ \langle \text{E} \rangle \\
\gamma_4 & : & \langle \text{E} \rangle & \rightarrow \quad \oplus\ \langle \text{E} \rangle \\
\gamma_5 & : & \langle \text{E} \rangle & \rightarrow \quad \langle \text{E} \rangle \oplus \langle \text{E} \rangle \\
\gamma_6 & : & \langle \text{E} \rangle & \rightarrow \quad (\ \langle \text{E} \rangle\ ) \\
 & & & \\
\gamma_7 & : & \langle \text{A} \rangle & \rightarrow \quad \text{x} = \langle \text{E} \rangle \\
 & & & \\
\gamma_8 & : & \langle \text{S} \rangle & \rightarrow \quad \epsilon \\
\gamma_9 & : & \langle \text{S} \rangle & \rightarrow \quad \text{let}\ \langle \text{A} \rangle \\
\gamma_{10} & : & \langle \text{S} \rangle & \rightarrow \quad \text{let}\ \langle \text{A} \rangle\ \text{where}\ \langle \text{A} \rangle \\
 & & & \\
\gamma_{11} & : & \langle \text{P} \rangle & \rightarrow \quad \$ \\
\gamma_{12} & : & \langle \text{P} \rangle & \rightarrow \quad \langle \text{S} \rangle\ \langle \text{P} \rangle
\end{array}
$$

Non-terminals are in ⟨angle brackets⟩.
"let" and "where" are each considered a single terminal. ϵ stands for the empty string and \$ stands for the end-of-stream marker. ⟨P⟩ is the start symbol. And we fix some specific operator $\oplus$ in the language. Each production rule is named differently for convenience, even if some of them may produce the same nonterminal. This grammar could very well describe a simple programming language!

Problem 1: Warmup (10 points)

- Give three derivations of the following string – one that is left-most, one that is right-most, and one that is neither.
  `let x =` $\oplus$ `x` $\oplus$ `x where x = x` $\oplus$ `x` $\oplus$ `x`

Problem 2: Predictive (LL(1)) Parsing (30 points)

(a) Compute the First and Follow sets for Λ.
Remember that First(γ) stands for the set of all tokens that may appear first in any string accepted by the production rule γ. Through a slight abuse of notation, First($\langle$N$\rangle$) can additionally be defined as the set of all tokens that may appear in any string from which the non-terminal $\langle$N$\rangle$ can be derived.

It is useful to compute the values of First, both for production rules and for non-terminals, because you can use them as intermediate values to compute other First and Follow sets. Therefore, you should write down both. Please show your work.

(b) Explain First-First conflicts (i.e. how this class of conflicts manifest themselves in the execution of a predictive parser), and how you would compute them using your answer to part (b). List all First-First conflicts you find in Λ.

(c) Explain First-Follow conflicts, and how you would compute them using your answer to part (b). List all First-Follow conflicts you find in Λ.

(d) Demonstrate the correct use of Left-Factoring and Right-Recursion to derive an equivalent grammar that resolves all conflicts you found in parts (c) and (d). Briefly explain why your transformations do not change the language.

Problem 3: LR Parsing (20 points)

(a) Explain shift-reduce and reduce-reduce conflicts (i.e. how they manifest themselves in an LR(1) parser).

(b) An LR(0) parser is a special case of LR parsers characterized by a zero token look-ahead. Reduce actions occur purely based on the tokens that have already been pushed on to the stack. List all the conflicts that an LR(0) parser for Λ would encounter.

(c) An SLR parser is an LR(0) parser with one addition: we perform a reduce action only if the next token is in the Follow set of the nonterminal that the reduction would derive. List all the conflicts that an SLR parser for Λ would encounter. Also explain what you would do to get the right-most possible derivations for Λ using an SLR parser if you had the ability to manually pick a state transition where there is a conflict.