CST Part IB Supervisions

Example Sheet 3

Petar Veličković

Lent Term 2016

Compiler Construction

This section of the examples will cover only parsing. The more advanced forms of parsing used to be a standalone topic on their own, and therefore I believe we should pay special attention to them.

1. Define the features of a *recursive descent parser*. When is using such a parser appropriate, and why would writing such a parser be desirable? What is the main potential pitfall of using such a parser?

2. Modify the following context-free grammar such that it would be suitable for parsing by recursive descent:

$$\begin{array}{lcl} S & \longrightarrow & P\ \$ \\ P & \longrightarrow & (T) \mid n \\ F & \longrightarrow & F * P \mid F\ /\ P \mid P \\ T & \longrightarrow & T + F \mid T - F \mid F \end{array}$$

   Afterwards, demonstrate its predictive parsing table. Show your working!

3. Define what it means for a grammar to be $LL(k)$ and $LR(k)$. Which of these two kinds of grammar is more expressive, and why?

4. Throughout this exercise*, the following $LR(1)$ grammar will be considered:

$$\begin{array}{lcl} S & \longrightarrow & E\ \boxed{\texttt{eof}} \\ E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & P * T \mid P \\ P & \longrightarrow & (E) \mid i \end{array}$$

   a) Construct the DFA associated with the $LR(0)$ states of this grammar. Can you then immediately conclude that this is not an $LR(0)$ grammar? If so, how?

*This exercise is the first and hopefully the only truly tedious one. I **strongly** urge you to complete it—it is very helpful for being completely certain you've truly understood the advanced parsing methods!

b) Derive the `First[X]` and `Follow[X]` sets for all nonterminal symbols X.

c) Use all of the above to derive the `action[s, t]` and `goto[s, X]` tables of the grammar, for all states s, terminals t and nonterminals X. You may list only the nonempty fields of the tables, if more convenient.

d) Demonstrate the steps taken when parsing `i + i` `eof`.

Practical work

For this week, there will be no programming assignment. If you would like to prepare for the next task of constructing a (recursive descent) parser for our toy language, you may wish to convert the syntax description of the language into a form that has no left recursion, and present a suitable data structure/type for encoding the parse tree.

These will be formally given as practical (pre-)exercises next week, but you may feel free to submit them this week, if you want to.