HW 2 – Unification, Regular Expressions, Parse Trees, Ambiguous Grammars, LRParsing

CS 421 – Fall 2011

Revision 1.0

Assigned Tuesday, October 25, 2011
Due Tuesday, November 1, 2011, 2:00 PM - in class
Extension No extension, due to proximity to second midterm

1 Change Log

1.1 Corrected a typo in Problem 4: "go to state 5" should have been "go to state 1". Added a clarification to Problem 2.b.

1.0 Initial Release.

2 Turn-In Procedure

Your answers to the following questions are to be hand-written, or printed, neatly on one or more sheets of paper, each with your name in the upper right corner. The homework is to be turned in in class at the start of class. Alternately, you may hand it to Prof. Elsa Gunter in person before the deadline.

3 Objectives and Background

The purpose of this HW is to test your understanding of

- How to unify a system of equations
- How to use regular expressions and regular grammars to formally express sets of strings (called *languages*) given by an English language description
- How to create a parse tree for a given string with a given grammar
- How to disambiguate a grammar
- How to write a recursive descent parser for an LL(1) grammar

Another purpose of HW2 is to provide you with experience answering non-programming written questions of the kind you may experience on the second midterm and final.
Caution: It is strongly advised that you know how to do these problems before the second midterm.

4 Problems

1. (16 points) Use the unification algorithm described in class and in MP6 to give a most general unifier, if one exists, for the following set of equations (unification problem). Capital letters (A, B, C, D, E) denote variables of unification. The lower-case letters (f, l, n, p) are constants or term constructors. (f and p have arity 2 - i.e., take 2 arguments, l has arity 1, and n has arity 0 - i.e. it is a constant.) Show all your work by listing the operations performed in each step of the unification and the result of that step. Your final answer should be a single simultaneous substitution, together with the result of applying the substitution to the given unification problem.

$$\{(f(p(A, f(D, B)), n)\ ,\ f(p(f(p(B, C), l(C)), A), C))\}$$

2. (16 points) For each of the following languages (*i.e.*, sets of strings), write a regular expression and a regular grammar generating the set:

   a. (6 points) The set of all strings of `a`'s, `b`'s, and `c`'s such that there exists exactly one `c`, before that `c` there is at least one character, but after that `c` there can be any number of characters.

   b. (8 points) The set of all strings representing lists of binary numbers, where the list separator is a semicolon and the list delimiters are the open on closed square bracket. The semicolon may only occur between two binary numbers occurring consecutively in the list. Each binary number may not have a leading `0` unless it is just `0`.

3. (23 points) Consider the following grammar over the terminal alphabet `0`, `1`, `2`, $+$, $-$, $=$, (,), `if`, `then`, `else`, with nonterminal and start symbol $<$ `num` $>$:

$$\begin{array}{rcl} <\texttt{num}> & ::= & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid <\texttt{num}> + <\texttt{num}> \mid <\texttt{num}> - <\texttt{num}> \mid (<\texttt{num}>) \mid \\ & & \texttt{if}\ <\texttt{num}> = <\texttt{num}>\ \texttt{then}\ <\texttt{num}>\ \texttt{else}\ <\texttt{num}> \end{array}$$

   a. (9 points)Show that the above grammar is ambigous by showing at least three distinct parse trees for the string

```
if 1 = 0 then 2 else 1 + 0 + 2
```

   b. (9 points) Disambiguate this grammar such that $+$ and $-$ have the same precidence and associate to the left, and the scope of the `if_then_else` extends as far to the right as is syntactically possible.

   c. (5 points) Give a parse for the string

```
if 1 = 0 then 2 else 1 + 0 + 2
```

   in the grammar you gave in the previous part of this problem.

4. (17 points) Given the following grammar over nonterminal `<m>`, `<e>` and `<t>`, and terminals `z`, `o`, `l`, `r`, `p` and `eof`, with start symbol `<m>`:

$$\begin{array}{ll} P0: & <\texttt{m}> ::= <\texttt{e}>\ \texttt{eof} \\ P1: & <\texttt{e}> ::= <\texttt{t}> \\ P2: & <\texttt{e}> ::= <\texttt{t}>\ \texttt{p}\ <\texttt{e}> \\ P3: & <\texttt{t}> ::= \texttt{z} \\ P4: & <\texttt{t}> ::= \texttt{o} \\ P5: & <\texttt{t}> ::= \texttt{l}\ <\texttt{e}>\ \texttt{r} \end{array}$$

   and Action and Goto tables generated by YACC for the above grammar:

State	Action							Goto		
	z	o	l	r	p	[eof]		<m>	<e>	<t>
st1	**s3**	**s4**	**s5**	**err**	**err**	**err**			**st2**	**st7**
st2	**err**	**err**	**err**	**err**	**err**	**a**				
st3	**r3**	**r3**	**r3**	**r3**	**r3**	**r3**				
st4	**r4**	**r4**	**r4**	**r4**	**r4**	**r4**				
st5	**s3**	**s4**	**s5**	**err**	**err**	**err**			**st8**	**st7**
st6	**err**	**err**	**err**	**err**	**err**	**a**				
st7	**err**	**err**	**err**	**r1**	**s9**	**r1**				
st8	**err**	**err**	**err**	**s10**	**err**	**err**				
st9	**s3**	**s4**	**s5**	**err**	**err**	**err**			**st11**	**st7**
st10	**r5**	**r5**	**r5**	**r5**	**r5**	**r5**				
st11	**r2**	**r2**	**r2**	**r2**	**r2**	**r2**				

where **st**i is state i, **s**i abbreviates **shift** i, **r**i abbreviates **reduce** i, **a** abbreviates **accept** and `[eof]` means we have reached the end of input, describe how the string `lzpor[eof]` would be parsed with an LR parser using these productions and tables by filling in the table on the next page. I have given you the first 5 cells in the first two rows to get you started. You will need to add more rows.

Stack	Current String	Action to be taken
Empty	`lzpor[eof]`	Initialize stack, go to state 1
st1	`lzpor[eof]`	

5. (Extra Credit) (10 points total)

   a. (4pts) Using the rule for polyomorphic typing given in class, give a type derivation for

   $$\{\texttt{x : int}\} \vdash \texttt{let id = fun x => x in id id x : int}$$

   Caution: no credit will be given for incorrect parsing!

   b. (6pts) Using the rules for monomorphic typing, prove that there does not exist a type τ such that

   $$\{\texttt{x : int}\} \vdash \texttt{let id = fun x => x in id id x : } \tau$$

   is valid. I expect a rigorous proof.