CS421: Progrmg Languages & Compilers
Summer 2015

Homework 7

University of Illinois, Urbana-Champaign,

Due: July 28, 2015, 5.00 pm. Submit PDF + code files on Moodle.

1 Theoretical Questions (50 points)

1.1 Objectives and Background

The purpose of this HW is to test your understanding of

- 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
- The difference between structural operational semantics and transition semantics.
- How to create rules for structural operational semantics.
- How to write rules for transition semantics.

Another purpose of homework is to provide you with experience answering non-programming written questions of the kind you may experience on the second midterm and final.

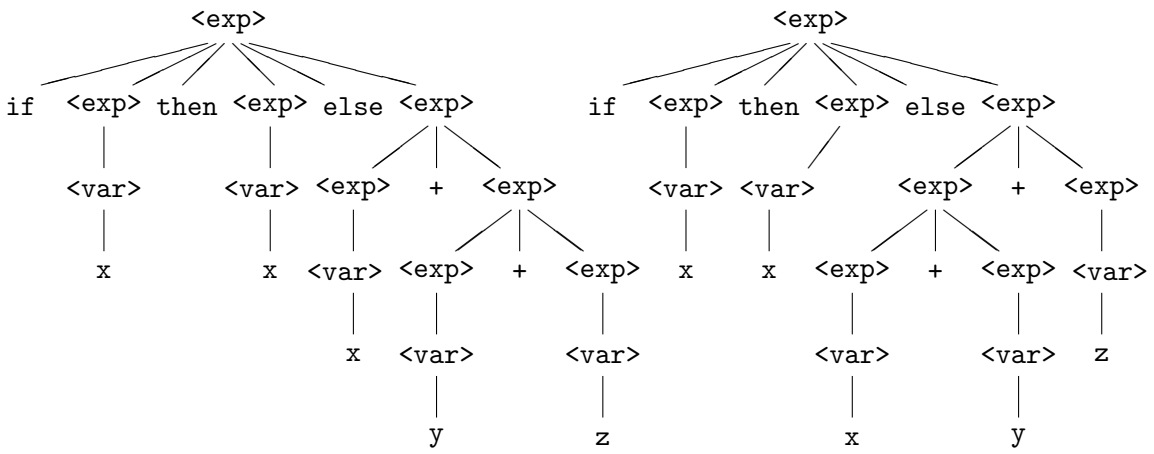
1.2 Problems

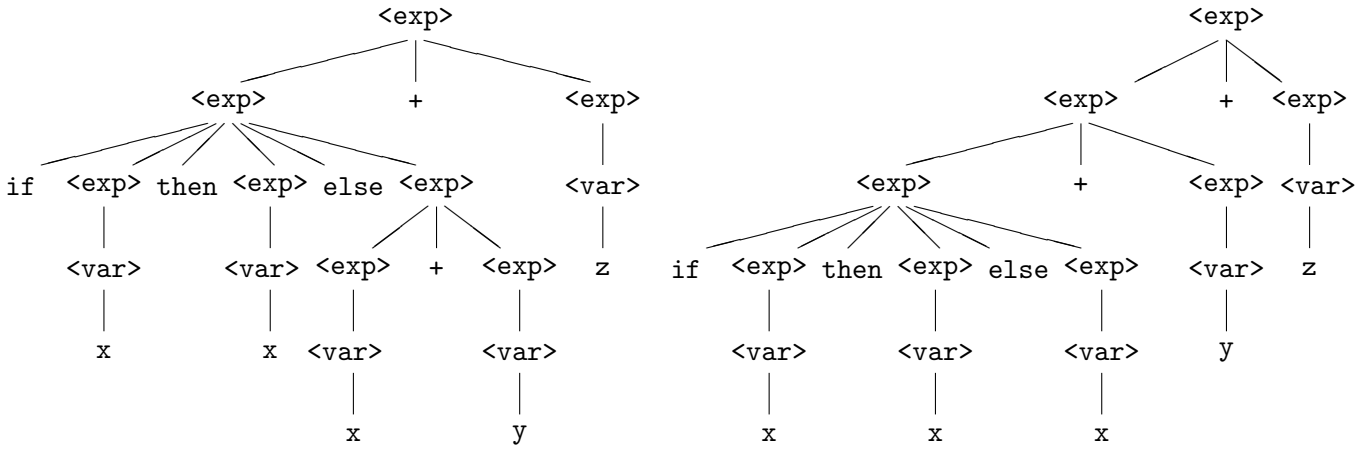
Problem 1: (22 points)

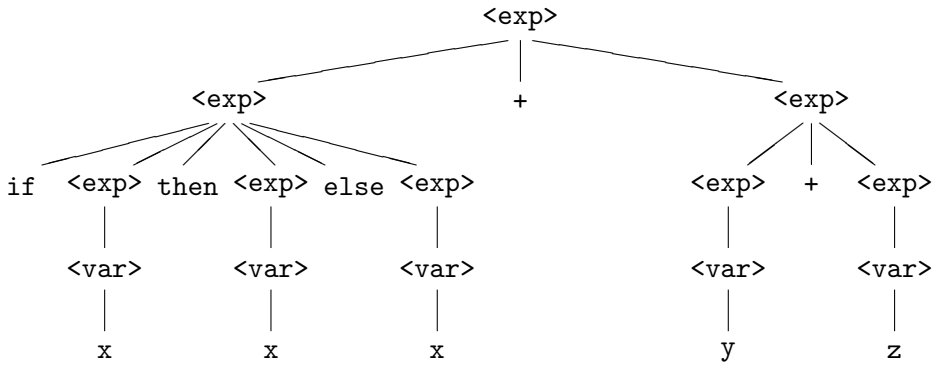
Consider the following grammar over the alphabet {`if`, `then`, `else`, +, `x`, `y`, `z`, (,)}:

$$
\begin{array}{lll}
\langle \texttt{exp} \rangle & ::= & \langle \texttt{var} \rangle \\
& & \mid \texttt{if}\ \langle \texttt{exp} \rangle\ \texttt{then}\ \langle \texttt{exp} \rangle\ \texttt{else}\ \langle \texttt{exp} \rangle \\
& & \mid \langle \texttt{exp} \rangle + \langle \texttt{exp} \rangle \\
& & \mid (\langle \texttt{exp} \rangle) \\
\langle \texttt{var} \rangle & ::= & \texttt{x} \mid \texttt{y} \mid \texttt{z}
\end{array}
$$

a. (9 points) Show that the above grammar is ambiguous by showing at least three distinct parse trees for the string "`if x then x else x + y + z`"
Solution: There are five possible parse trees for `if x then x else x + y + z` with the given grammar. Yours should be three among them.







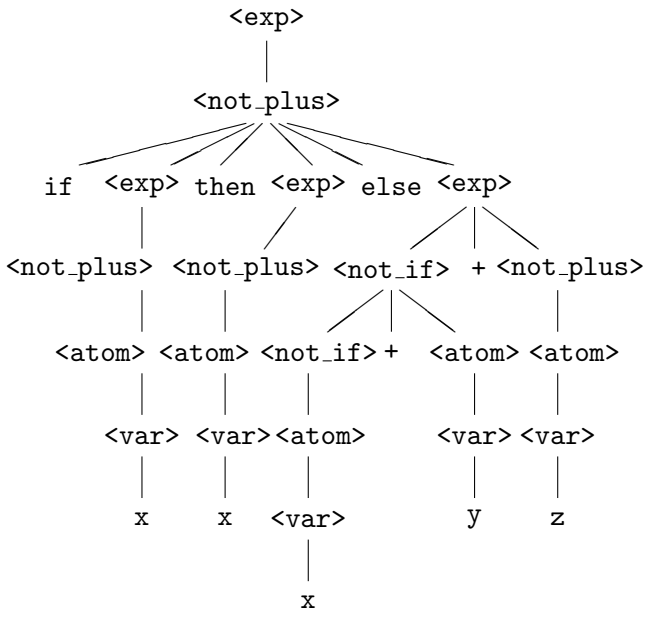
b. (10 points) Write a new grammar accepting the same language that is unambiguous, and such that addtion ⟨exp⟩ + ⟨exp⟩ has higher precedence than conditional

`if` ⟨exp⟩ `then` ⟨exp⟩ `else` ⟨exp⟩ , and such that addition associates to the left.

Solution:

$$
\begin{array}{rcl}
< \texttt{exp} > & ::= & < \texttt{not\_if} > + < \texttt{not\_plus} > \mid < \texttt{not\_plus} > \\
< \texttt{not\_if} > & ::= & < \texttt{not\_if} > + < \texttt{atom} > \mid < \texttt{atom} > \\
< \texttt{not\_plus} > & ::= & \texttt{if} < \texttt{exp} > \texttt{then} < \texttt{exp} > \texttt{else} < \texttt{exp} > \mid < \texttt{atom} > \\
< \texttt{atom} > & ::= & < \texttt{var} > \mid (< \texttt{exp} >) \\
< \texttt{var} > & ::= & \texttt{x} \mid \texttt{y} \mid \texttt{z}
\end{array}
$$

c. (3 points) Give the parse tree for "`if x then x else x + y + z`" using the grammar you gave in the previous part of this problem.
Solution:



Problem 2: (28 points)

Consider the following language syntax.
$I \in Identifiers$
$N \in Numerals$
$B ::= \texttt{true} \mid \texttt{false} \mid B \,\&\, B \mid B \texttt{ or } B \mid \texttt{not } B \mid E < E \mid E = E$
$E ::= N \mid I \mid E + E \mid E * E \mid E - E \mid -E$
$C ::= \texttt{skip} \mid C; C \mid I \texttt{::=} E \mid \texttt{if } B \texttt{ then } C \texttt{ else } C \texttt{ fi} \mid \texttt{while } B \texttt{ do } C \texttt{ od}$

Add a new increment operator `++I` to the syntax of expression `E` and a new do-while operator `do` C `while` B `od` to the syntax of commands `C`.

a. (10 points) Add the structural operational semantics (*a.k.a.* natural semantics) for these operators. Note that the operators work as follows. The semantics of the operator `++I` is to add one to the current value of I, then store the new value into I.

The execution of `do` C `while` B `od` starts with executing the command C in the body of the loop. The loop is repeated until the boolean expression B is evaluated to false.

Solution:

$$\frac{v = m(I) + 1}{(++I, m) \Downarrow (v, m[I \leftarrow v])}$$

$$\frac{(C, m) \Downarrow m' \quad (B, m') \Downarrow \texttt{true} \quad (\texttt{do } C \texttt{ while } B \texttt{ od}, m') \Downarrow m''}{(\texttt{do } C \texttt{ while } B \texttt{ od}, m) \Downarrow m''}$$

$$\frac{(C, m) \Downarrow m' \quad (B, m') \Downarrow \texttt{false}}{(\texttt{do } C \texttt{ while } B \texttt{ od}, m) \Downarrow m'}$$

b. (10 points) Add the transition semantics for these operators. They have the same meaning as part a.

Solution:

$$\frac{v = m(I) + 1}{(++I, m) \rightarrow (v, m[I \leftarrow v])}$$

$$\frac{}{(\texttt{do } C \texttt{ while } B \texttt{ od}, m) \rightarrow (C; \texttt{if } B \texttt{ then do } C \texttt{ while } B \texttt{ od else skip fi}, m)}$$

c. (8 points) Using the rules given for natural semantics in class, and the rules written in parts a and b, give a proof that starting with a memory that maps `x` to 3, `do y` $::=$ $++$`x while x` $<$ `5 od` evaluates to a memory that maps `x` and `y` to 5.

Solution:

Let $\mathcal{M}_3 = \{\texttt{x} \rightarrow \texttt{3}\}$, $\mathcal{M}_4 = \{\texttt{y} \rightarrow \texttt{4}, \texttt{x} \rightarrow \texttt{4}\}$ and $\mathcal{M}_5 = \{\texttt{y} \rightarrow \texttt{5}, \texttt{x} \rightarrow \texttt{5}\}$. Also, let A stand for assignment, R for relational operator, I for the increment we added, DW for do while loops we added.

$$\dfrac{\dfrac{\dfrac{\texttt{4} = \texttt{3} + \texttt{1}}{(++\texttt{x}, \mathcal{M}_3) \Downarrow (\texttt{4}, \{\texttt{x} \rightarrow \texttt{4}\})}\,\text{I}}{(\texttt{y} ::= ++\texttt{x}, \mathcal{M}_3) \Downarrow \mathcal{M}_4}\,\text{A} \quad \dfrac{\dfrac{\textit{Identifier}}{(\texttt{x}, \mathcal{M}_4) \Downarrow (\texttt{4}, \mathcal{M}_4)} \quad \dfrac{\textit{Numeral}}{(\texttt{5}, \mathcal{M}_4) \Downarrow (\texttt{5}, \mathcal{M}_4)} \quad \texttt{4} < \texttt{5} = \texttt{true}}{(\texttt{x} < \texttt{5}, \mathcal{M}_4) \Downarrow (\texttt{true}, \mathcal{M}_4)}\,\text{R} \quad \textsc{SubTree1}}{(\texttt{do y} ::= ++\texttt{x while x} < \texttt{5 od}, \mathcal{M}_3) \Downarrow \mathcal{M}_5}\,\text{DW}$$

where SubTree1 is given by

$$\dfrac{\dfrac{\dfrac{\texttt{5} = \texttt{4} + \texttt{1}}{(++\texttt{x}, \mathcal{M}_4) \Downarrow (\texttt{5}, \{\texttt{y} \rightarrow \texttt{4}, \texttt{x} \rightarrow \texttt{5}\})}\,\text{I}}{(\texttt{y} ::= ++\texttt{x}, \mathcal{M}_4) \Downarrow \mathcal{M}_5}\,\text{A} \quad \dfrac{\dfrac{\textit{Identifier}}{(\texttt{x}, \mathcal{M}_5) \Downarrow (\texttt{5}, \mathcal{M}_5)} \quad \dfrac{\textit{Numeral}}{(\texttt{5}, \mathcal{M}_5) \Downarrow (\texttt{5}, \mathcal{M}_5)} \quad \texttt{5} < \texttt{5} = \texttt{false}}{(\texttt{x} < \texttt{5}, \mathcal{M}_5) \Downarrow (\texttt{false}, \mathcal{M}_5)}\,\text{R}}{(\texttt{do y} ::= ++\texttt{x while x} < \texttt{5 od}, \mathcal{M}_4) \Downarrow \mathcal{M}_5}\,\text{DW}$$

Note:
The solutions have a pair of value and memory for the expressions instead of just a value to

account for the added increment operator. However, for this part, no points will be deducted for just returning a value if it is correct. That is, a modified memory must be shown for the increment operator but a plain value for relational operators, identifiers and numerals is acceptable.

Uncredited Problem 1: (0 points)

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

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

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

	Action							Goto		
State	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]`	

Solution:

Stack	Current String	Action to be taken
Empty	`lzpor[eof]`	Initialize stack, go to state 1
st1	`lzpor[eof]`	shift `l`, go to state 5
st1::l::st5	`zpor[eof]`	shift `z`, go to state 3
st1::l::st5::z::st3	`por[eof]`	reduce by rule 3, go to state 7
st1::l::st5::<t>::st7	`por[eof]`	shift `p`, go to state 9
st1::l::st5::<t>::st7::p::st9	`or[eof]`	shift `o`, go to state 4
st1::l::st5::<t>::st7::p::st9::o::st4	`r[eof]`	reduce by rule 4, go to state 7
st1::l::st5::<t>::st7::p::st9::<t>::st7:	`r[eof]`	reduce by rule 1, go to state 11
st1::l::st5::<t>::st7::p::st9::<e>::st11	`r[eof]`	reduce by rule 2, go to state 8
st1::l::st5::<e>::st8	`r[eof]`	shift `r`, go to state 10
st1::l::st5::<e>::st8::r::st10	`[eof]`	reduce by rule 5, go to state 7
st1::<t>::st7	`[eof]`	reduce by rule 1, go to state 2
st1::<e>::st2	`[eof]`	accept

2 Machine Problems (50 points)

Please see `solution.mll`