

Solutions for Sample Questions for Midterm 2 (CS 421 Fall 2008)

On the actual midterm, you will have plenty of space to put your answers.
Some of these questions may be reused for the exam.

1. Using the rules provided in class, derive a valid typing judgment for
let rec fact = fun n -> if n = 0 then 1 else let r = fact (n - 1) in n * r in fact;;
 (The rules will be provided for you on the exam, if this kind of question is asked.)

Solution: (I didn't just write the tree because it wouldn't fit.)

By the let_rec rule we have

$$\frac{\begin{array}{l} (1) \{ \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{fun } n \rightarrow \text{if } n = 0 \text{ then } 1 \text{ else let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int} \rightarrow \text{int} \\ (2) \{ \text{fact} : \text{int} \rightarrow \text{int} \} \vdash \text{fact} : \text{int} \rightarrow \text{int} \end{array}}{\{ \} \vdash \text{let rec fact = fun } n \rightarrow \text{if } n = 0 \text{ then } 1 \text{ else let } r = \text{fact } (n - 1) \text{ in } n * r \text{ in fact} : \text{int} \rightarrow \text{int}}$$

(2) is valid by the variable rule:

$$\frac{}{(2) \{ \text{fact} : \text{int} \rightarrow \text{int} \} \vdash \text{fact} : \text{int} \rightarrow \text{int}}$$

By the fun rule we have

$$\frac{(3) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{if } n = 0 \text{ then } 1 \text{ else let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int}}{(1) \{ \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{fun } n \rightarrow \text{if } n = 0 \text{ then } 1 \text{ else let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int} \rightarrow \text{int}}$$

By the if_then_else rule we have

$$\frac{\begin{array}{l} (4) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (n = 0) : \text{bool} \quad (5) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash 1 : \text{int} \\ (6) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int} \end{array}}{(3) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{if } n = 0 \text{ then } 1 \text{ else let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int}}$$

(5) is valid by the rule for constants. By the rule for binary relations we have

$$\frac{\begin{array}{l} (7) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash n : \text{int} \quad (8) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash 0 : \text{int} \end{array}}{(4) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (n = 0) : \text{bool}}$$

(7) is valid by the rule for variables. (8) is valid by the rule for constants.

By the rule for let, we have

$$\frac{\begin{array}{l} (9) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash \text{fact } (n - 1) : \text{int} \quad (10) \{ r : \text{int}, n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (n * r) : \text{int} \end{array}}{(6) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (\text{let } r = \text{fact } (n - 1) \text{ in } n * r) : \text{int}}$$

By the rule for applications we have

$$\frac{\begin{array}{l} (11) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash \text{fact} : \text{int} \rightarrow \text{int} \quad (12) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash (n - 1) : \text{int} \end{array}}{(9) \{ n : \text{int}, \text{fact} : \text{int} \rightarrow \text{int} \} \vdash \text{fact } (n - 1) : \text{int}}$$

(11) is valid by the variable rule. By the rule for binary operations, we have

$$\frac{(13)\{n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash n : \text{int} \quad (14)\{n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash 1 : \text{int}}{(12)\{n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash (n - 1) : \text{int}}$$

(13) is valid by the variable rule. (14) is valid by the constant rule. Thus (12) is valid. Thus (9) is valid. We have (10) left. By the rule for binary operations we have:

$$\frac{(15) \{r:\text{int}, n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash n : \text{int} \quad (16) \{r:\text{int}, n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash r : \text{int}}{(10) \{r:\text{int}, n:\text{int}, \text{fact}:\text{int} \rightarrow \text{int}\} \vdash (n * r) : \text{int}}$$

(15) and (16) are both valid by the variable rule. Hence (10) is valid. Hence (6) is valid. Hence (3) is valid, and hence (1) is valid

2. Give a (most general) unifier for the following unification instance. Capital letters denote variables of unification. Show your work by listing the operation performed in each step of the unification and the result of that step.

$$X = f(g(x),W), h(y) = Y, f(Z,x) = f(Y,W)\}$$

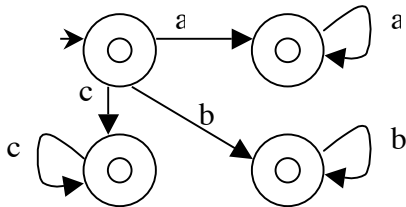
Solution:

- {X = f(g(x),W), h(y) = Y, f(Z,x) = f(Y,W)}
- {h(y) = Y, f(Z,x) = f(Y,W)} with {X = f(g(x),W)} by eliminate
- {Y = h(y), f(Z,x) = f(Y,W)} with {X = f(g(x),W)} by orient
- {f(Z,x) = f(h(y),W)} with {X = f(g(x),W), Y = h(y)} by eliminate
- {Z = h(y), x=W} with {X = f(g(x),W), Y = h(y)} by decompose
- {x = W} with {X = f(g(x),W), Y = h(y), Z = h(y)} by eliminate
- {W = x} with {X = f(g(x),W), Y = h(y), Z = h(y)} by orient
- answer: {X = f(g(x),x), Y = h(y), Z = h(y), W = x} by eliminate

3. For each of the regular expressions below (over the alphabet {a,b,c}), draw a non-deterministic finite state automaton that accepts exactly the same set of strings as the given regular expression.

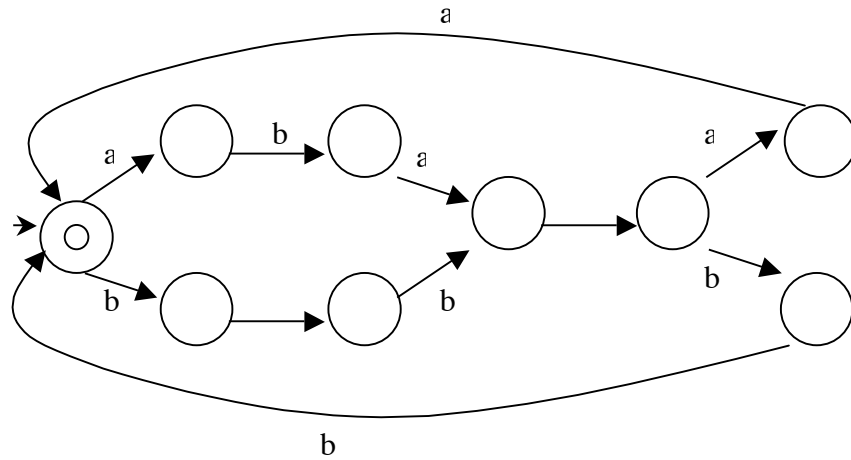
a. $a^* \vee b^* \vee c^*$

Solution:



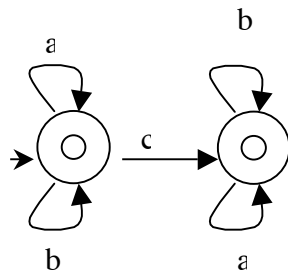
b. $((aba \vee bab) c (aa \vee bb))^*$

Solution:



c. $(a^*b^*)^*(c \vee \epsilon) (b^*a^*)^*$

Solution:



4. Consider the following grammar:

$\langle S \rangle ::= \langle A \rangle \mid \langle A \rangle \langle S \rangle$

$\langle A \rangle ::= \langle Id \rangle \mid (\langle B \rangle$

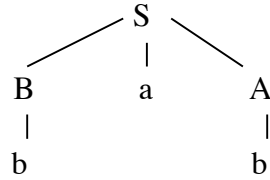
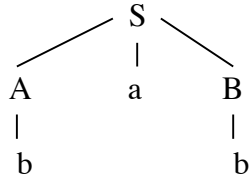
$\langle B \rangle ::= \langle Id \rangle \mid \langle Id \rangle \langle B \rangle \mid (\langle B \rangle$

$\langle Id \rangle ::= 0 \mid 1$

For each of the following strings, give a parse tree for the following expression as an $\langle S \rangle$, if one exists, or write "No parse" otherwise:

Give an example of a string for which this grammar has two different parse trees, and give its parse trees.

Solution: String: bab



6. Write an unambiguous grammar generating the set of all strings over the alphabet $\{0, 1, +, -\}$, where $+$ and $-$ are infix operators which both associate to the left and such that $+$ binds more tightly than $-$.

Solution:

```

<S> ::= <plus> | <S> - <plus>
<plus> ::= <id> | <plus> + <id>
<id> ::= 0 | 1
  
```

7. Write a recursive descent parser for the following grammar:

```

<S> ::= <N> % <S> | <N>
<N> ::= g <N> | a | b
  
```

You should include a datatype **token** of tokens input into the parser, one or more datatypes representing the parse trees produced by parsing (the abstract syntax trees), and the function(s) to produce the abstract syntax trees. Your parser should take a list of tokens as input and generate an abstract syntax tree corresponding to the parse of the input token list.

Solution:

```

type token = ATk | BTk | GTk | PercentTk
type s = Percent of (n * s) | N_as_s n
and n = G of n | A | B
  
```

```

let rec s_parse tokens =
  match n_parse tokens with (n, tokens_after_n) ->
    (match tokens_after_n with PercentTk::tokens_after_percent ->
      (match s_parse tokens_after_percent
        with (s, tokens_after_s) -> (Percent (n,s), tokens_after_s)
         | _ -> (N_as_s n, tokens_after_n))
    and n_parse tokens =
  match tokens
  with GTk::tokens_after_g ->
    (match n_parse tokens_after_g
  
```


where si is shift and stack state i , ry is reduce using production P_j , acc is accept. The blank cells should be considered as labeled with error. The empty “character” represents end of input

Describe how the sentence $fx*y<eol>$ would be parsed with an LR parser using this table. For each step of the process give the parser action (shift/reduce), input and stack state

Solution: In the table below, the top of the stack is on the left

Curr State	Current Stack :	Curr String	Action
		$fx*y<eol>$	Init stack and state
st1	st1	$fx* y<eol>$	Shift f to stack, go to state 5
st5	st1 : f : st5	$x*y<eol>$	Shift x, go to state 3
st3	st1 : f : <u>st5</u> : x : st3	$*y<eol>$	Reduce by prod 5: $N \Rightarrow x$, ie remove st3 and x from the stack, temporarily putting us in st5, push N and st8 (because $GOTO(st5, N) = st8$) onto stack go to state 8
st8	<u>st1</u> : f : <u>st5</u> : N : <u>st8</u>	$*y<eol>$	Reduce by prod 4: $N \Rightarrow fN$, go to state 7
st7	<u>st1</u> : N : <u>st7</u>	$*y<eol>$	Reduce by prod 3: $E \Rightarrow N$, go to state 6
st6	st1 : E : st6	$*y<eol>$	Shift *, go to state 9
st9	st1 : E : st7 : * : st9	$y<eol>$	Shift y, go to state 4
st4	st1 : E : st7 : * : <u>st9</u> : y : <u>st4</u>	$<eol>$	Reduce by prod 6: $N \Rightarrow y$, go to state 11
st12	<u>st1</u> : E : <u>st7</u> : * : <u>st9</u> : N : <u>st11</u>	$<eol>$	Reduce by prod 2: $E \Rightarrow E*N$, go to state 6
st7	st1 : E : st6	$<eol>$	Shift $<eol>$, go to state 10
st11	<u>st1</u> : E : st6 : $<eol>$: <u>st10</u>		Reduce by prod 1: $S \Rightarrow E<eol>$, go to state 2
st2	st1 : S : st2		Accept