

Solutions for Sample Questions for Midterm 2 (CS 421 Spring 2007)

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. Write the definition of an OCaml variant type **reg_exp** to express abstract syntax trees for regular expressions over a base character set of booleans. Thus, a boolean is a **reg_exp**, epsilon is a **reg_exp**, the concatenation of two **reg_exp**'s is a **reg_exp**, the "choice" of two **reg_exp**'s is a **reg_exp**, and the Kleene star of a **reg_exp** is a **reg_exp**.

Solution:

```
type reg_exp =  
  Epsilon  
| Var of bool  
| Choice of (reg_exp * reg_exp)  
| Concat of (reg_exp * reg_exp)  
| Kleene_star of reg_exp  
| Paren of reg_exp (* I would accept it with this case missing *)
```

2. Given the following OCAML datatype:

```
type int_seq = Null | Snoc of (int_seq * int)
```

write a tail-recursive function in OCAML **all_pos : int_seq -> bool** that returns **true** if every integer in the input **int_seq** to which **all_pos** is applied is strictly greater than 0 and **false** otherwise. Thus **all_pos (Snoc(Snoc(Snoc(Null,3),5),7))** should return **true**, but

all_pos (Snoc(Null,~1)) and **all_pos (Snoc(Snoc(Null, 3),0))** should both return **false**.

Solution:

```
let rec all_pos s =  
  (match s with Null -> true  
   | Snoc(seq, x) -> if x <= 0 then false else all_pos seq);;
```

3. Using the rules provided in class, derive a valid type 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:

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

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

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

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

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

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

By the rule for let, we have

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

By the rule for applications we have

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

(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

4. 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.

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

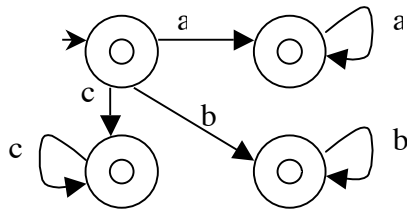
Solution:

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

5. For each of the regular expressions below (over the alphabet $\{a,b,c\}$), draw a nondeterministic finite state automaton that accepts exactly the same set of strings as the given regular expression.

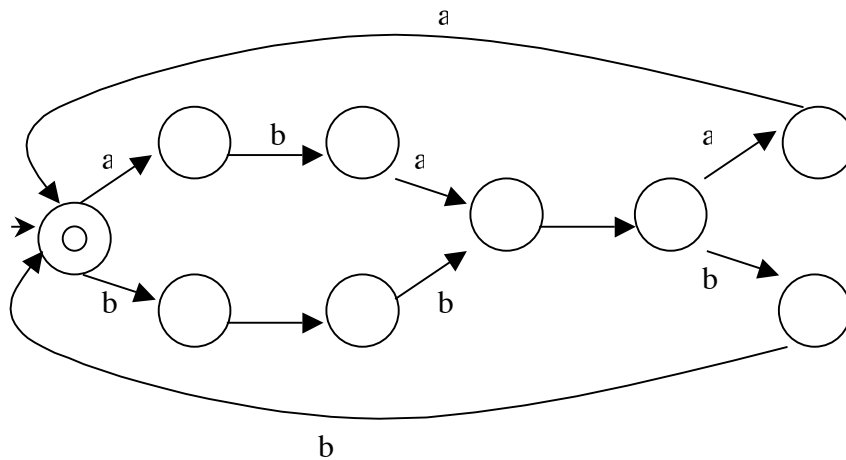
a. $a^* \vee b^* \vee c^*$

Solution:



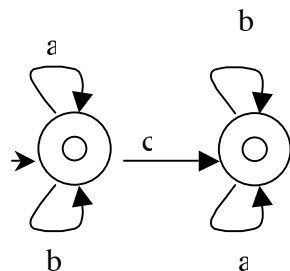
b. $((aba \vee bab) c (aa \vee bb))^*$

Solution:



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

Solution:



6. Consider the following grammar:

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

$\langle A \rangle ::= \langle \text{Id} \rangle \mid (\langle B \rangle$

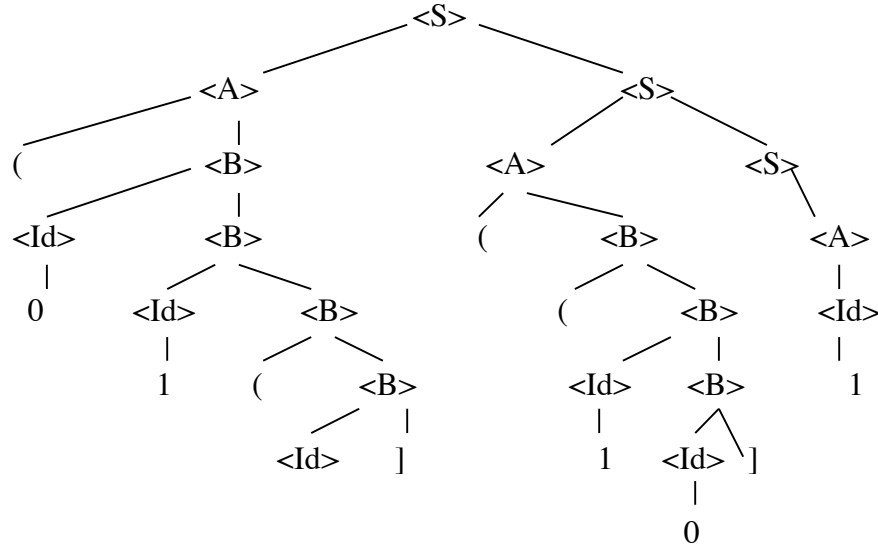
$\langle B \rangle ::= \langle \text{Id} \rangle \mid \mid \langle \text{Id} \rangle \langle B \rangle \mid (\langle B \rangle$

$\langle \text{Id} \rangle ::= 0 \mid 1$

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:

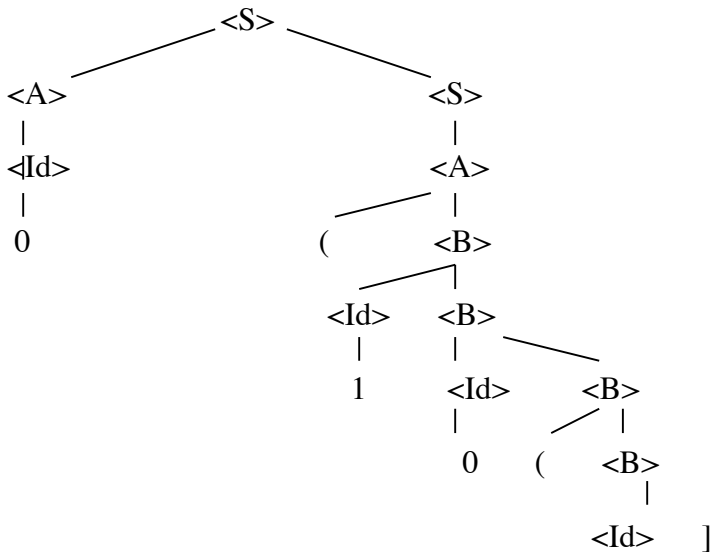
a. $(01(1]((10]1$

Solution:



b. $0(10(1]$

Solution:



c. $(0(101]0]$

Solution: No parse tree

7. Consider the following ambiguous grammar (Capitals are nonterminals, lowercase are terminals):

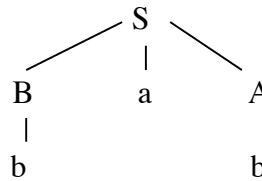
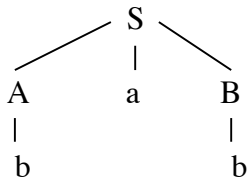
$$S \rightarrow A a B \mid B a A$$

$$A \rightarrow b \mid c$$

$$B \rightarrow a \mid b$$

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

Solution: String: bab



8. 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:

$$\langle S \rangle ::= \langle \text{plus} \rangle \mid \langle S \rangle - \langle \text{plus} \rangle$$

$$\langle \text{plus} \rangle ::= \langle \text{Id} \rangle \mid \langle \text{plus} \rangle + \langle \text{id} \rangle$$

$$\langle \text{id} \rangle ::= 0 \mid 1$$