CS411 Homework 7
Control-flow, logic programming. *Due: 4 May 2004*

Correction [4/25, 2pm]: typo in written problem 3: the label statement is "$l : c$", not "$l : s$".

Turn in all of the assignment using CMS. Turn in the written part in a file `written.txt` (or .doc, or .pdf), and submit your code in a file `solve.ml`. ..

1 Written part

1. Give a sample program fragment that produces different results for each of the following:

   (a) Call-by-value and static scoping;
   (b) Call-by-value and dynamic scoping;
   (c) Call-by-name and dynamic scoping;

2. (a) Write a translation of exceptions into a language with if expressions, let constructs, pairs, and selection operators for the first and second components of the tuple;
   (b) Extend the above translation to support constructs that pass values along with exceptions, e.g., "e_1 `handle` $X(x)$ `=>` e_2" and "`throw` $X(e)$".

3. We extend IMP with labels and goto statements:

$$\begin{array}{ll} \text{Commands} & c ::= \ldots \;\mid\; l : c \;\mid\; \texttt{goto}\ l \\ \text{Lables} & l \in \mathsf{Labels} \end{array}$$

   The jump command `goto` l transfers the control at the point where the label command $l : c$ occurs. For simplicity, assume that the label command occurs before the jump during the execution.

   We'd like to model configurations as tuples (s, m, k, c) containing: a store s, a map m from labels to continuations, a current continuation k, and a currently executed command c. Continuations can be simply represented as commands, so m maps labels to commands, and k is some command c'. The evaluation of assignments is:

$$\begin{array}{ll} (s, m, c, x := a) \to (s, m, c, x := a') & \text{if } \langle a, s\rangle \to \langle a', s\rangle \\ (s, m, c, x := v) \to (s[x \mapsto v], m, c, \texttt{skip}) & \end{array}$$

   Write the rules for `skip`, $c_1; c_2$, `if` b `then` c_1 `else` c_2, `while` b `do` c, $l : c$, and `goto` l. Note: make sure that while loops work correctly in the presence of goto's.

2 Programming part

In this part you will implement an interpreter for a mini-Prolog language called `plog`. A `plog` program consists of a set of rules (i.e., Horn clauses), followed by query that consists of a set of goals. A solution must satisfy all of the goals together. Here is an example program:

```
parent(bob, ann).
parent(bob, tom).
parent(bob, joe).
parent(amy, ann).
female(ann).
female(amy).
male(joe).
male(tom).
mother(X,Y) :- parent(X,Y), female(X).
father(X,Y) :- parent(X,Y), male(X).

? parent(X, Y), male(Y).
```

Terms and rules are represented using the following Ocaml types:

```
type term = Term of string * term list
          | Var  of string

type rule = term * term list
```

The program is a list of rules; and the query is a list of terms. Your job is to implement the function `solve` in file `solve.ml`:

```
solve:  (rule list) -> (term list) -> (term -> term) list
```

For a list of rule `r` and a set of query `q`, the call `solve r q` returns all the solutions to the query, i.e., the set of all unifiers that make `q` feasible. If the query is not feasible, the returned list is empty. The main program in `main.ml` calls `solve` and displays all the solutions. For instance, the result for the above program is:

```
Found 2 solution(s).

Solution:
Y = joe
X = bob

Solution:
Y = tom
X = bob
```

For your convenience, we have provided a functional implementation of union-find structures (or unifiers); and a function `match_rule r crt rest` that matches (via unification) a current goal `crt` with a rule `r`, for a remaining set of goals `rest`. The function `match_rule` returns the new set of goals if unification succeeds; or raises `FailUnify` otherwise.

Each time a rule is matched, all of its variables must be alpha-renamed. The function `fresh r` renames all variables in rule `r` and returns the renamed rule. Different calls `fresh r` for the same rule `r` must yield different renamings.

To do: Implement the unification function `unify` and the renaming function `fresh`, and then use `match_rule` to implement function `solve`. Turn in your solution in CMS, in file `solve.ml`.