CS232F: Programming Languages
II semester 2001-02

Programming Assignment 2: Computations as proofs.

Consider the following data type declarations for integer Let-expressions.

```
type var = string;
datatype IExpr = INT of int
               | VAR of var
               | PLUS of IExpr * IExpr
               | MINUS of IExpr * IExpr
               | MULT of IExpr * IExpr
               | DIV of IExpr * IExpr
               | MOD of IExpr * IExpr
               | LET of IExpr * IExpr * IExpr
   | ERR of IExpr
```

Now do the following:

- Define a data type **env** for environments (along with the necessary operations like environment updation).
- Define a data type **rule** (with side-conditions if necessary) along with names for each rule.
- Define the data type **proof** for proofs which will eventually be displayed as output. Each line of the proof also contains a justification which is a list of the names of the rules invoked.
- Now implement the **deterministic** small-step rules for left-to-right evaluation of expressions for this language of integer expressions as given in the class. Assume that all evaluations of expressions start with an *empty* environment.

THE USER INTERFACE
Input A user of your program gives an expression to your program in the syntax of the above data-type declaration. Hence a user specifies an expression for evaluation in the form

```
LET (VAR("x"), INT(5),
     LET (VAR("y"), MINUS (VAR("x"), INT(~2)),
          MULT (VAR("x"), PLUS (VAR("y"), VAR("x"))
         )
    )
```

which stands for the ML expression

```
let val x = 5
 in let val y = x - ~2
     in x * (y + x)
    end
end
```

Output The output may be one of the following:

1. A syntax error because the user's input does not conform to the syntax of the datatype of integer Let-expressions.
2. Use of undeclared variables in the expression. This has to be pointed out with the message along with the name of the undeclared variable.

UNDECLARED VARIABLE: <variable-name>

3. In the rare event (possibly by sheer accident!) if a user manages to give an input with no such errors, then a proof along with justification of each step in the proof (with a reference to the rule used) should be displayed for the benefit of the user. This proof acts also as the execution sequence for this simple language.

4. The output of a perfectly written expression of type `IExpr` is an element of type proof such that the concluding line of the proof is the unique normal form of the input expression.

5. An expression of type `IExpr` is in normal form if and only if it is of one of the following forms.

   (a) A value of type `int`, or

   (b) An expression of the form `ERR (e)`, where `e` is an expression of type `IExpr` without any occurrences of the tag `ERR` in it.

Notes

1. A "`LET`" expression should have its first argument of the form "`VAR(<string>)`" as in the case of ML let expressions.

2. Since there is no facility for defining functions or recursive definitions and no possibility of self-reference, no possibilities of infinite computations exist. Hence all computations are finite and hence all proofs are finite.

3. Since computations are proofs and they are all finite, what do they prove? Well, they prove the existence of normal forms and unique ways of obtaining them. A computation or a proof is merely the proof of existence (by "derivation") of the normal form.

4. Note that an expression may yield a normal form of the form `ERR(e)` only under one of the following conditions.

   (a) The given expression already has an occurrence of `ERR` in it, or

   (b) because some subexpression within `e` has a "division-by-zero" subexpression in it.

5. Note that `ERR(e)` is in normal form if and only if there are no other occurrences of `ERR` within the expression `e`.

6. As part of the language definition, no user is allowed to use an undeclared variable.

7. Note that the following kinds of expressions are also allowed

```
LET (VAR("a"), LET (VAR("b"), INT(5),
                PLUS(VAR("b"), INT(1))
               ),
     MULT(VAR("a"), VAR("a"))
    )
```

   The above expression should yield the normal form `36`.

8. Since the rules of the language (as given in the class) are deterministic, every valid let-exprression in the language has a single proof which yields a value of type int, or an error.

9. You need to give a set of **deterministic** rules for error detection ("division-by-zero"). You also need to make sure that normal forms are <u>*unique*</u> and every expression has a unique normal form obtained by a <u>*unique proof*</u>![1]

[1]This is a major deviation from whatever you have done in the last eighteen years of your life, where it didn't seem to matter whether any thing was deterministic at all. You thought there were several ways of getting the same answer, and it didn't matter as long as you got the right answer.