CSE 130: Programming Languages – Principles and Paradigms Fall 2000

Problem Set 3

Instructor: Daniele Micciancio Nov. 1, 2000

Due: Nov 13, beginning of class. No late submissions accepted.
Guidelines for the programming assignments: In this problem set you will use ML to write your programs. The subset of ML studied in class should be enough to solve all of the problems. In particular, you should not use the "imperative" features of ML. For all programs you are required to type them and run them using the sml interactive system. Together with your solutions you should submit a print out of the programs, and the result of executing them. You should also include a brief english description of how your programs work. Clarity of your programs (as well as the english description) is as important as correctness. In particular, in order to get full credit your programs should be properly formatted so to make them easy to read. No electronic copy of your programs is required.
Collaboration policy: As usual, you should do your homeworks individually. See collaboration policy on the web page for details.

Problem 1 (6 points)

(a) For each of the following expressions, give the corresponding ML type:

- `(("1",[2,3]),4.5)`
- `(([abc"],4),[nil,[5,6,7]])`

(b) For each of the following types, give a corresponding ML expression:

- `int -> ((string*int) list)`
- `((int*int)*real*(bool list))*(real*string)`

[Note: some of the parentheses in part (b) and (c) have been added for clarity. ML might not print some of the redundant parentheses.]

(c) For each of the following polymorphic types, define a corresponding ML function:

- `'a*'a*string -> 'a`
- `'a*'b*('a list * 'b list) -> ('a list * 'b list)`

Problem 2 (10 points)

(a) Define an ML recursive datatype `'label Tree23` to represent trees in which each internal node has either 2 or 3 children. Both internal nodes and leaves can contain an optional label of type 'label. (You can use ML predefined `'a option` datatype.) [Note: All trees have at least one node. You do not have to define a constructor for the empty tree.]

(b) Define a polymorphic function `frontier` of type `'a Tree23 -> 'a list`, that on input a tree returns the list of the labels stored at the leaves of the tree. (You can use ML infix append operator `@` to concatenate lists.)

Test your function on two simple `int Tree23` and `string Tree23` of your choice.

(c) If in part (b) you used `@`, try to find a more efficient solution (without `@`) that uses an auxiliary function `front-aux: ('a Tree23)*('a list) -> ('a list)`.

Problem 3 (14 points)

In this problem you are asked to write a set of programs for the symbolic differentiation and evaluation of rational expressions. Part (a) and (b) are independent of each other and can be solved in any order. Part (c) requires the solution of part (a) and (b) first.

(a) Define an abstract datatype `Rat` to represent rational numbers that supports the following operations:

- Addition (`Add: Rat * Rat -> Rat`), compute the sum of two rationals
- Multiplication (`Mul: Rat * Rat -> Rat`), compute the product of two numbers
- Division (`Div: Rat * Rat -> Rat`), compute the quotient of two numbers. If a division by zero occurs, then an exception `DivByZero` should be raised.
- Test for zero (`IsZero: Rat -> bool`), test if a number is zero
- Conversion from integer (`Int2Rat: int -> Rat`), convert an integer into the corresponding rational number.
- Conversion to integers (`Rat2Ints: Rat -> int*int`), convert a rational into a pair of integers corresponding to the numerator and denominator of the fraction. This function should be used only for printing the value of a rational number.

Your solution should look something like this:

```
local
    fun gcd(a,0) = a
      | gcd(a,b) = gcd(b,a mod b);
in
    abstype Rat = ...
    with
        exception DivByZero;
        fun Int2Rat ...;
        fun Add ...;
        fun Mul ...;
        fun Div ...;
        fun IsZero ...;
        fun Rat2Ints ...;
```

```
    end;
end;
```

[Note: `gcd` is a function defined locally to the abstract datatype that computes the greatest common divisor of two integers. You might find it useful to "simplify" fractions.]

(b) Consider the following datatype to represent rational expressions in a single variable `X` (i.e., expressions built from a variable symbol `X` and integer constants, using addition, product and division).

```
datatype Exp = X | Const of int | Sum of Exp*Exp
             | Prod of Exp*Exp | Frac of Exp*Exp;
```

Define an ML function `derive:Exp->Exp` that on input expression `E:Exp`, returns expression `(derive E): Exp` corresponding to the symbolic derivative of `E` with respect to `X`.

For example on input $(x+1)x$, your function should output $(1+0)*x+(x+1)*1$ or some equivalent expession (simplifying the result is not required):

```
- derive(Prod(Sum(X,Const(1)),X));
val it = Sum(Prod(Sum(Const(1),Const(0)),X),
             Prod(Sum(X,Const(1)),Const(1))): Exp
```

Test your program on this, as well as two other simple expressions of your choice. In your print out, the "truncated" expressions as given by ML are acceptable.

(c) In this last part, you have to define an evaluation function that on input an expression `E:Exp` and an integer `n : int`, returns the value of `E` at `X=n`, possibly trying to handle `DivByZero` exceptions. Break up your solution as follows.

First define a simple evaluation function `simpleEval:Exp*int->Rat`, that on input expression `E` and integer `n`, computes the value of `E` at `n`, without trying to deal with possible exceptions that can arise during the computation (i.e., exceptions are not caught and percolate to the top level).

Then, use functions `derive` and `simpleEval` from the previous parts to define a more elaborate evaluation function `eval:Exp*int->Rat` that tries to handle exceptions as follow. If the evaluation of `E` at `n` raises a `DivByZero` exception, then transform `E` into an equivalent expression of the form `Frac(E1,E2)` where `E1` and `E2` are polynomial expressions, i.e. expressions that do not use `Frac`. [You can use the ML function `Rationalize` posted on the course webpage to perform this transformation.] If both `E1` and `E2` evaluate to 0, tries to evaluate the limit of `Frac(E1,E2)` when `X` tends to n using Hopital's Rule:

$$\lim_{x \to n} \frac{f(x)}{g(x)} = \frac{f'(n)}{g'(n)}.$$

In other words, your program should evaluate the derivatives of `E1` and `E2` at n, and compute the quotient of the two numbers.

Your function should return a rational number or raise one of the following two exceptions

```
exception Infinity;
exception Indefinite;
```

as appropriate. I.e., if $E2(n) \neq 0$ then the result should be $E1(n)/E2(n)$. If $E1(n) \neq 0$ and $E2(n) = 0$, then your program should raise `Infinity`. If both $E1(n) = 0$ and $E2(n) = 0$, then the result depends on the value of $E1'(n)$ and $E2'(n)$. If $E2'(n)$ is not zero, then the result should be $E1'(n)/E2'(n)$. Otherwise (i.e., if $E2'(n) = 0$), your program should raise either `Infinity` (if $E1'(n) \neq 0$), or `Indefinite` (if $E1'(n) = 0$).

Test your program on expression

$$\frac{x - 1/x}{1 - 1/x}$$

or, equivalently, in ML syntax:

```
Frac(Sum(X,Frac(Const(~1),X)),
     Sum(Const(1),Frac(Const(~1),X)))
```

[Notice: in ML the unary minus operator is $\sim$.] You should compute the expression at $X = 1$, first using `simpleEval`, and then using `eval`. [Use `Rat2Ints` to display rational numbers.]