CSE 130: Programming Languages – Principles and Paradigms Fall 2000

Problem Set 3 - Solutions

Instructor: Daniele Micciancio Nov. 15, 2000

Problem 1 (6 points)

(a)

- `(("1",[2,3]),4.5): (string * int list) * real;`
- `((["abc"],4),[nil,[5,6,7]]): (string list * int) * int list list;`

(b)

- `fn x=>if x=1 then [("a",x)] else [("b",x)] : int -> ((string*int) list)`
- `(((1,1),1.1,[true]),(1.1,"1")): ((int*int)*real*(bool list))*(real*string);`

(c)

- `fn (x,y,z)=> if z="a" then x else y : 'a*'a*string -> 'a`
- `fn (x,y,z,t)=>(x::z,y::t) : 'a*'b*('a list * 'b list) -> ('a list * 'b list)`

Problem 2 (10 points)

(a) The following is a type specification for 2-3 Trees. We use different constructors for the two types of internal nodes of such trees. It is also correct to use the *option* type to describe nodes with a third optional child.

```
datatype 'label Tree23 =
         L of 'label option |
         N2 of 'label * 'label Tree23 * 'label Tree23 |
         N3 of 'label * 'label Tree23 * 'label Tree23 * 'label Tree23;
```

(b) A function that computes the frontier of a 23 Tree can be recursively described as follows: if the tree has only one leaf with a possibly empty label, then the result should be a list containing that particular label. If the root is a node which has some children, the result should be the concatenation of the frontiers of its children:

```
fun frontier(L(NONE))=[] |
    frontier(L(SOME l))=[l]|
    frontier(N(l,T1,T2,NONE))=frontier(T1)@frontier(T2) |
    frontier(N(l,T1,T2,SOME T3))=frontier(T1)@frontier(T2)@frontier(T3);
```

```
frontier( N3 ("1",
             N2("2",
                 L(SOME "c"),
                 L(SOME "s")
                ),
             N2("3",
                 L(SOME "e"),
                 L(SOME "1")
                ),
             N2("4",
                 L(SOME "3"),
                 L(SOME "0")
                )
              )
          );
```

returns `val it = ["c","s","e","1","3","0"] : string list`
and

```
frontier( N3 (1,
             N2(2,
                 L(SOME 3),
                 L(SOME 4)
                ),
             N2(5,
                 L(SOME 6),
                 L(SOME 7)
                ),
             N2(4,
                 L(SOME 8),
                 L(SOME 9)
                )
              )
          );
```

returns `val it = [3,4,6,7,8,9] : int list`

(c) Function `frontier` that doesn't use `@`:

```
fun frontier2(T)=
let fun frontaux(L(NONE),l)=l |
        frontaux(L(SOME label),l)=label::l |
        frontaux(N2(label,T1,T2),l)= frontaux(T1,frontaux(T2,l)) |
        frontaux(N3(label,T1,T2,T3),l)=frontaux(T1, frontaux(T2, frontaux(T3,l)));
```

```
in
    frontaux(T,[])
end;
```

Problem 3 (14 points)

(a)

```
local
    fun gcd(a,0) = a
      | gcd(a,b) = gcd(b,a mod b);
in
    abstype Rat = Q of int*int
    with
        exception DivByZero;
        fun Int2Rat(x) = Q(x,1);
        fun Add (Q(p1,q1),Q(p2,q2)) =
            let val p = p1*q2+q1*p2;
                val q = q1*q2;
                val g = gcd(p,q)
            in Q(p div g,q div g)
            end;
        fun Mul (Q(p1,q1),Q(p2,q2)) =
            let val p = p1*p2;
                val q = q1*q2;
                val g = gcd(p,q)
            in Q(p div g,q div g)
            end;
        fun Div (Q(p1,q1),Q(p2,q2)) =
            let val p = p1*q2;
                val q = q1*p2;
                val g = gcd(p,q)
            in if q=0 then
                raise DivByZero
               else Q(p div g,q div g)
            end;
        fun IsZero (Q(p,q)) = (p = 0);
        fun Rat2Ints (Q(p,q)) = (p,q);
    end;
end;
```

(b)

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

fun Rationalize(X) = Frac(X,Const(1))
  | Rationalize(Const(x)) = Frac(Const(x),Const(1))
  | Rationalize(Sum(E1,E2)) =
    let val Frac(E1p,E1q) = Rationalize(E1);
        val Frac(E2p,E2q) = Rationalize(E2)
    in Frac(Sum(Prod(E1p,E2q),Prod(E2p,E1q)),Prod(E1q,E2q))
    end
  | Rationalize(Prod(E1,E2)) =
    let val Frac(E1p,E1q) = Rationalize(E1);
        val Frac(E2p,E2q) = Rationalize(E2)
    in Frac(Prod(E1p,E2p),Prod(E1q,E2q))
    end
  | Rationalize(Frac(E1,E2)) =
    let val Frac(E1p,E1q) = Rationalize(E1);
        val Frac(E2p,E2q) = Rationalize(E2)
    in Frac(Prod(E1p,E2q),Prod(E2p,E1q))
    end;

fun derive(X) = Const(1)
  | derive(Const(x)) = Const(0)
  | derive(Sum(E1,E2)) = Sum(derive(E1),derive(E2))
  | derive(Prod(E1,E2)) = Sum(Prod(E1,derive(E2)),Prod(E2,derive(E1)))
  | derive(Frac(E1,E2)) =
    Frac(Sum(Prod(derive(E1),(E2)),
             Prod(Const(~1),Prod(derive(E2),E1))),
         Prod(E2,E2));
```

(c)

```
fun simpleEval(X,n) = Int2Rat(n)
  | simpleEval(Const(x),n) = Int2Rat(x)
  | simpleEval(Sum(E1,E2),n) = Add(simpleEval(E1,n),simpleEval(E2,n))
  | simpleEval(Prod(E1,E2),n) = Mul(simpleEval(E1,n),simpleEval(E2,n))
  | simpleEval(Frac(E1,E2),n) = Div(simpleEval(E1,n),simpleEval(E2,n));

exception Infinity;
exception Indefinite;
```

```
fun eval(E,n) =
    simpleEval(E,n)
    handle DivByZero =>
        let val Frac(E1,E2) = Rationalize(E);
            val V1 = simpleEval(E1,n);
            val V2 = simpleEval(E2,n)
        in if (not(IsZero(V2))) then Div(V1,V2)
           else if (not(IsZero(V1))) then raise Infinity
                else
                    let val V1 = simpleEval(derive(E1),n);
                        val V2 = simpleEval(derive(E2),n)
                    in if (not(IsZero(V2))) then Div(V1,V2)
                       else raise if (not(IsZero(V1))) then Infinity
                            else Indefinite
                    end
        end;

val E = Frac(Sum(X,Frac(Const(~1),X)),Sum(Const(1),Frac(Const(~1),X)));
```

When one tries to compute `Rat2Ints(simpleEval(E,1));` there will be an uncought exception `DivByZero`. On the other hand, if one uses `Rat2Ints(eval(E,1));`, the exception will be treated, namely there will be one application of L'Hospital rule, and the result will be: `val it = (2,1) : int * int`