Homework 7

SNU 4190.310, 2006 가을

이 광근

Due: 12/18(Mon), 24:00

Exercise 1 (40pts) "저지방 고단백 M"

M 실행기 위에, 지난 숙제에서 구현한 단순 타입 시스템(simple type system)을 대신해서 보다 정교한 let-다형 타입 시스템(let-polymorphic type system)을 장착하자. 그래서, 단순 타입 시스템이 받아들이는 프로그램은 물론이고, 다형 타입의 함수(polymorphic function, 다양한 타입의 인자에 적용될 수 있는 함수)를 가지는 프로그램까지 받아들여 지도록.

예를들어, 아래와 같은 잘 도는 프로그램들이 단순 타입 시스템에서는 받아들여지지 않았으나, let-다형 타입 시스템에서는 받아들여지게 된다.

TA가 제공하는 M 실행기의 틀 위에 let-다형 타입 시스템을 장착하라.

```
(* example 1: polymorphic toys *)

   let val I = fn x => x
       val add = fn x => x.1 + x.1
       val const = fn n => 10
   in
     I I;
     add(1, true) + add(2, "snu 310 fall 2006");
     const 1 + const true + const "kwangkeun yi"
   end

(* example 2: polymorphism with imperatives *)

   let val f = fn x => malloc x
   in
      let val a = f 10
```

```
          val b = f "pl"
          val c = f true
      in
        a := !a + 1;
        b := "hw7";
        c := !c or false
      end
   end

(* example 3: polymorphic swap *)

   let val swap =
           fn order_pair =>
              if order_pair.1 order_pair.2
               then order_pair.2
               else (order_pair.2.2, order_pair.2.1)
   in
     swap(fn pair => pair.1 + 1 = pair.2, (1,2));
     swap(fn pair => pair.1 or pair.2, (true, false))
   end

(* S K I combinators *)

   let val I = fn x => x
       val K = fn x => fn y => x
       val S = fn x => fn y => fn z => (x z) (y z)
   in
    S (K(SI)) (S(KK)I) 1 (fn x => x+1)
   end
```

□

Exercise 2 (40pts) "Algebraic Data Type"

위의 프로그래밍 시스템은 재귀적으로 정의되는 데이타(inductive data)를 다루는 함수들을 잘 받아들이지 못한다.

예를 들어, 리스트(list)나 이진 트리(binary tree)는 다음과 같이 프로그램하게 될 것이지만, 우리의 let-다형 타입 시스템은 이 프로그램들을 받아들이지 못한다:

```
(* program with list *)

let val nil = 0
    val isNil = fn list => list = 0
```

```
    val link = fn element => fn list => (element, list)
    val first = fn list => list.1
    val rest = fn list => list.2

in
   let
     val list = link 2 (link 1 (link 0 nil))
     rec sum = fn list => if isNil list then 0
                          else first list + sum (rest list)
   in
     sum list
   end
end

(* program with binary tree *)

let val emptyTree = 0
   val leaf = fn n => (n, (emptyTree, emptyTree))
   val node = fn n => fn ltree => fn rtree => (n, (ltree, rtree))
   val value = fn tree => tree.1
   val leftTree = fn tree => tree.2.1
   val rightTree = fn tree => tree.2.2
   val isEmpty = fn tree => tree = emptyTree
   val isLeaf = fn tree => isEmpty(leftTree tree) and isEmpty(rightTree tree)
in
   let
     rec traverse =
          fn tree => if isEmpty tree then 0
                     else if isLeaf tree then value tree
                     else value tree + traverse (leftTree tree)
                          + traverse (rightTree tree)
     val tree = node 8 (leaf 7) (node 10 (leaf 9) emptyTree)
   in
     traverse tree
   end
end
```

어떻게 언어를 확장하고, 타입 시스템은 어떻게 하면 될 런지, 논의하라. □