Homework 8

SNU 4190.310, 2015 가을

Kwangkeun Yi

Program Due: 12/15(Tue), 24:00

Report Due: 12/16(Wed), 10:00

이번 숙제의 목적은:

- M언어로 짜여진 프로그램이 무난히 실행될 수 있는지를 미리 확인해 주는 안전장치를 갖추어 본다.
- let-다형 타입 시스템(let-polymorphic type system)을 장착해서 편리하면서도 안전한 프로그래밍 환경을 완성해 본다. (프로그램으로 제출)
- 무슨 함수가 어디에서 호출되는 지를 미리 예측하는 예측기를 고안해 본다. (리포트를 302-428호 IN Box에 제출)

Exercise 1 (80pts) "저지방 고단백 M"

M 실행기 위에, let-다형 타입 시스템(let-polymorphic type system)을 장착하자. 예를들어, 아래와 같은 잘 도는 프로그램들을 생각하자. 단순 타입 시스템은 받아들이지 않는 프로그램들이다. 하지만 장착할 let-다형 타입 시스템은 모두 받아들여야 할 프로그램들이다.

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");
    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 (S I)) (S (K K) I) 1 (fn x => x+1)
  end
```

□

Exercise 2 (80pts) "람다예보"

최근의 프로그래밍 언어들(ML, Scala, Haskell, Python, JavaScript, C++14, C#, F# 등)은 람다계산법같이 함수를 자유롭게 다룰 수 있게 해준다. 함수가 여타 다른 데이터와 다르지 않다.

이 숙제는 그런 프로그램의 실행중에 무슨 일이 일어날 지를 미리 자동으로 예측하는 도구를 고안하는 것이다. 특히, 그런 프로그램의 핵심만을 대상으로 하자. 함수가 자유롭게 계산되는 언어의 핵심을 대상으로 무슨 함수가 어디에서 호출되는 지를 예측하는 프로그램이다.[1]

아래의 적극적인(eager-evaluation, or call-by-value) 프로그래밍 언어를 생각하자. 수업에서 다룬 언어의 일부다.

$$
\begin{array}{rcll}
e & ::= & n & \text{integer} \\
 & | & x & \text{identifier} \\
 & | & \lambda x.e & \text{function} \\
 & | & f\ \lambda x.e & \text{recursive function} \\
 & | & e\ e & \text{application} \\
 & | & \texttt{ifz}\ e\ e\ e & \text{branch} \\
 & | & e \texttt{+} e & \text{addtion}
\end{array}
$$

프로그램에는 함수식들이 산재해 있는데, 우리가 예측할 것은 함수호출식마다 어떤 함수가 호출될 지를 예측하는 것이다. 예를들어 아래 식을 보면 다섯 개의 람다식이 있다.

$$(\lambda f.(\lambda x.(f\ 0)\ (x\ 10)))\ (\lambda y.(\lambda x.x + y)\ (\lambda z.z)$$

위의 식에서 함수호출식은 세 개가 있는데 ($f\ 0$, $x\ 10$, $(f\ 0)\ (x\ 10)$) 이 식들이 어떤 함수를 호출하게 될 지를 미리 예측하는 것이다.

일반적으로 프로그램의 모든 식마다 그 식이 계산 결과로 내놓게되는 함수들의 집합을 예측하면 될 것이다. 그 집합들에 관한 방정식을 세우고 풀 수 있으면 된다.

- 방정식의 변수(unknown): 주어진 프로그램의 모든 식마다 번호를 붙이고 프로그램 변수들은 모두 다르다고 하자. 방정식의 변수는 다음과 같다. 변수 X_i 는 i번 식의 실행결과로 나올 함수식들의 집합이다. 변수 X_x는 프로그램 변수 x가 가지게 될 함수식들의 집합이다.

[1]우리가 만드는 물건이 생각대로 작동할 지를 미리 확인하려는 욕구는 모든 공학분야에 공통적이다. 따라서 프로그램 실행을 미리 예측하려는 것은 특별한 것이 아니다. 다른 공학이나 과학과 달리 그 대상이 프로그램이라는 것 뿐이다. 자연의 다이나믹스를 예측하는 물리학, 혹은 기계장치와 전기장치의 다이나믹스를 예측하는 기계공학과 전기공학과 같다. 기계설계를 하고 미리 제대로 작동할 지 예측하기 위해 우리는 미분방정식을 푼다. 우리는 프로그램을 대상으로 그 다이나믹스(실행상황)를 미리 예측하는 방법을 실현해 보는 것이다.

- 연립 방정식을 세우는 규칙: 다음의 규칙을 써서 프로그램식을 한번 훑으면 방정식이 모여진다. 방정식을 다음과 같은 꼴로 표현하게된다.

$$\mathcal{X} \supseteq setexp$$

$$\begin{array}{rcl} setexp & ::= & \{\lambda x.e\} \\ & | & \mathcal{X} \\ & | & \mathcal{X} \,@\, \mathcal{Y} \\ & | & \mathcal{X} \cup \mathcal{Y} \end{array}$$

  위에서 $\mathcal{X}, \mathcal{Y}$는 방정식 변수 $X_1, X_2, \cdots$등을 뜻한다.

  프로그램 e의 연립 방정식 E를 세우는 ($e \vdash E$ 라고 쓰자) 규칙을 e의 경우에 맞추어 재귀적으로 정의한 일부는 다음과 같다. 빈칸과 나머지 경우를 모두 정의해서 제출하라.

$$\frac{}{n_i \vdash X_i \supseteq \emptyset} \qquad \frac{}{x_i \vdash X_i \supseteq \boxed{\cdots}}$$

$$\frac{e_k \vdash E}{(\lambda x.e_k)_i \vdash (X_i \supseteq \{\lambda x.e_k\}) \ \wedge \ E} \qquad \frac{e_k \vdash E}{(f\ \lambda x.e_k)_i \vdash (X_i \supseteq \{\lambda x.e_k\}) \ \wedge \ \boxed{\cdots} \ \wedge \ E}$$

$$\frac{e_m \vdash E_1 \qquad e_n \vdash E_2}{(e_m\ e_n)_i \vdash (X_i \supseteq \boxed{\cdots}) \ \wedge \ E_1 \ \wedge \ E_2}$$

- 방정식 풀기: 모인 방정식을 풀면 된다. 푸는 방법은 아래의 규칙을 적용할 수 있는한 계속 적용해서 방정식들을 모아가는 것이다.

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \qquad \mathcal{Y} \supseteq \{\lambda x.e\}}{\mathcal{X} \supseteq \{\lambda x.e\}} \qquad \frac{\mathcal{X} \supseteq \mathcal{Y} \,@\, \mathcal{Z} \qquad \mathcal{Y} \supseteq \{\lambda x.e_i\}}{X_x \supseteq \mathcal{Z} \qquad \mathcal{X} \supseteq X_i}$$

  첫번째 규칙은 알게 된 사실을 단순히 전파하는 규칙이다. 두번째 규칙은 함수호출식의 경우 알게 된 사실을 전파하는 것이다. X가 Y-호출-Z의 결과를 가지게 되고 ($X \supseteq Y \,@\, Z$), Y가 어떤 함수인지를 알게되면 ($Y \supseteq \{\lambda x.e_i\}$), 함수 인자 X_x는 Z을 가지게 되고 ($X_x \supseteq Z$) 호출식 결과 X는 그 함수의 몸통식 e_i의 결과 X_i를 가지게 된다 ($X \supseteq X_i$).

- 방정식의 해: 위의 방식으로 모두 모아가면 언젠가는 끝난다. 프로그램에서 함수식의 갯수는 유한하기 때문이다. 방정식 변수 X_i의 답은 위와 같이 모은 방정식중에서 다음과 같은 명백한 꼴들을 모두 모으면 된다.

$$X_i \supseteq \{\lambda x.e_1\} \quad X_i \supseteq \{\lambda x.e_2\} \quad \cdots$$

  오른쪽에 있는 함수식들의 집합이 식 e_i가 실행중에 가지는 함수식들을 모두 포함하게 된다. □