

Homework 8

SNU 4190.310, 2023 봄

Kwangkeun Yi

Due: 6/14(수), 24:00

이번 숙제의 목적은:

- M언어로 짜여진 프로그램이 무난히 실행될 수 있는지를 미리 확인해 주는 정교한 안전장치를 갖추어 보기. 즉, let-다형 타입 시스템(let-polymorphic type system)을 장착해서 편리하면서도 안전한 프로그래밍 환경을 완성해 보기.
- 애플사에서 자랑하는 하위언어 자동번역 기술을 손수 이뤄보기. (특별한 기술이 아님을 겪고 자신감을 가지기)
- 타입시스템과는 다른 종류의 프로그램 실행 성질을 미리 예측해보는 도구를 디자인해보기. (이런것도 가능하겠구나, 를 겪어보기)

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

```

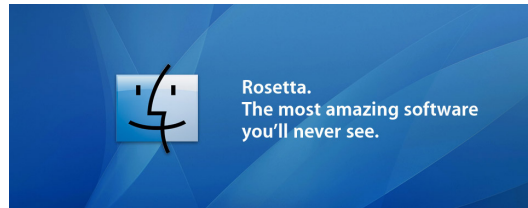
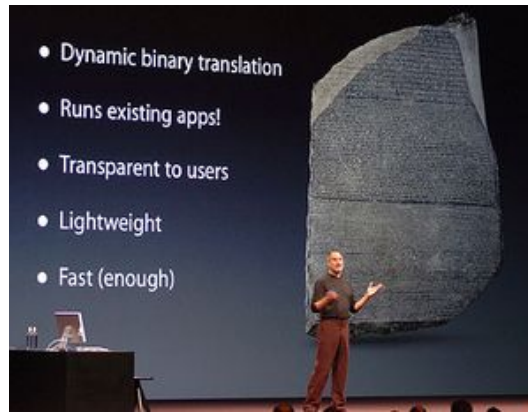
□

Exercise 2 (40pts) “SM5 RosettaX”

오늘날과 같은 Apple사의 발전에 핵심역할을 한 소프트웨어 중 하나가 기계어 번역(binary translation) 기술(정확하게는 “실행중 기계어 번역”(dynamic binary

translation) 기술) 이었다.

202X년 부터 애플은 제품에 사용하던 Intel 프로세서를 자사 프로세서 Mx로 바꿀예정이다. (여기서부터는 가설) 202X년 Apple World Wide Developer Conference에서 애플은 RosettaX 라는 소프트웨어 기술을 소개할 것이다. 소비자가 새로운 애플제품을 구매해도 Intel 프로세서에서 작동하던 예전 소프트웨어를 Mx 기반 제품에서 아무 문제없이 쓸 수 있게 해주는 기술이다. 이 RosettaX 는 2005년 애플이 PowerPC에서 Intel로 바뀔때 개발한 Rosetta 소프트웨어의 새 버전이다.



Rosetta 소프트웨어의 엔진은 애플이 “Transitive” 라는 회사



에서 구입한 기계어 번역기였다. 당시 Transitive사는 영국 맨체스터에 본사를 둔 아주 작은 벤처회사 였다.¹

이번 속제는 이러한 번역기 RosettaX를 SM5에 대해서 만들어보는 것이다.

¹Transitive사는 그 후 IBM에 인수됩니다. 당연히 돈방석에 앉았겠지요, 그 회사 연구원들과 설립자인 맨체스터 대학의 Alasdair Rawsthorne교수등.

SM5 프로그램을 Sonata 프로그램으로 번역하는 번역기이다.

Sonata가 SM5와 다른 점은 다음과 같다. 이외에는 모두 같다.

- Sonata는 SM5 부품 중에서 K (“continuation”) 부품이 없다:

$$(S, M, E, C).$$

따라서, SM5에서 K 부품을 건드렸던 `call` 명령의 의미와 빈명령문의 의미는 Sonata에서는 다음과 같다:

$$\begin{aligned} & (l :: v :: (x, C', E') :: S, \quad M, \quad E, \quad \text{call} :: C) \\ \Rightarrow & (S, \quad M\{l \mapsto v\}, \quad (x, l) :: E', \quad C') \\ & (S, \quad M, \quad E, \quad \text{empty}) \\ \Rightarrow & (S, \quad M, \quad E, \quad \text{empty}) \end{aligned}$$

- 메모리는 스택에 저장할 수 있는 값들을 모두 저장할 수 있다. 즉,

$$\begin{aligned} S & \in \text{Stack} = \text{Svalue list} \\ M & \in \text{Memory} = \text{Loc} \rightarrow \text{Svalue} \end{aligned}$$

여러분이 고안할 것은, 문제없이 돌아가는 SM5 코드를 받아서 Sonata 코드로 변환하는 함수

```
rosettax: Sm5.command -> Sonata.command
```

를 작성하는 것입니다. 모듈 `Sm5`와 모듈 `Sonata`는 제공됩니다. □

Exercise 3 (30pts. PDF파일로 제출) “람다예보”

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

이 속제는 그런 프로그램의 경우, 실행중에 무슨 일이 일어날 지를 미리 자동으로 예측하는 도구를 고안하는 것이다. 특히, 그런 프로그래밍 언어의 핵심만을 대상으로해서, 무슨 함수가 어디에서 호출되는 지를 예측하는 도구(프로그램)을 고안하자. 이런 도구를 통해서, 우리가 짠 프로그램이 실행중에 혹시나 우리가 생각하지 못한 행동을 할 지를 확인하는 데 도움을 받게 된다.²

²우리가 만드는 물건이 생각대로 작동할 지를 미리 확인하려는 욕구는 모든 공학분야에 공통적이

아래의 적극적인(eager-evaluation, or call-by-value) 프로그래밍 언어를 생각 하자. 수업에서 다른 언어의 일부다.

$e ::=$	n	integer
	$ $ x	identifier
	$ $ $\lambda x.e$	function
	$ $ $f \lambda x.e$	recursive function
	$ $ ee	application
	$ $ ifz eee	branch
	$ $ $e + e$	addtion

프로그램에는 함수식들이 산재해 있는데, 우리가 예측할 것은 함수호출식마다 어떤 함수가 호출될 지를 예측하는 것이다. 예를들어 아래 식을 보면 다섯 개의 랩다식이 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 i 번 식의 실행결과로 나올 함수식들의 집합이다. 변수 X_x 는 프로그램 변수 x 가 가지게 될 함수식들의 집합이다.

- 연립 방정식을 세우는 규칙: 다음의 규칙을 써서 프로그램식을 한번 훑으면

다. 따라서 프로그램 실행을 미리 예측하려는 것은 특별한 것이 아니다. 다른 공학이나 과학과 달리 그 대상이 프로그램이라는 것 뿐이다. 자연의ダイ나믹스를 예측하는 물리학, 혹은 기계장치와 전기 장치의 다이내믹스를 예측하는 기계공학과 전기공학과 같다. 기계설계를 하고 미리 제대로 작동할 지 예측하기 위해 우리는 미분방정식을 푼다. 우리는 프로그램을 대상으로 그 다이내믹스(실행상황)를 미리 예측하는 방법을 실현해 보는 것이다.

방정식이 모여진다. 연립 방정식 E 는 다음과 같은 꼴로 표현한다.

$$\begin{aligned}
 E & ::= \mathcal{X} \supseteq \text{setexp} \\
 & \quad | E \wedge E \\
 \text{setexp} & ::= \emptyset \\
 & \quad | \{\lambda x.e\} \\
 & \quad | \mathcal{X} \\
 & \quad | \mathcal{X} @ \mathcal{Y} \\
 & \quad | \mathcal{X} \cup \mathcal{Y}
 \end{aligned}$$

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

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

$$\begin{array}{c}
 \frac{}{n_i \vdash X_i \supseteq \emptyset} \qquad \frac{}{x_i \vdash X_i \supseteq \boxed{\dots}} \\
 \\
 \frac{e_k \vdash E}{(\lambda x.e_k)_i \vdash (X_i \supseteq \{\lambda x.e_k\}) \wedge E} \qquad \frac{e_k \vdash E}{(f \lambda x.e_k)_i \vdash (X_i \supseteq \{\lambda x.e_k\}) \wedge \boxed{\dots} \wedge E} \\
 \\
 \frac{e_m \vdash E_1 \quad e_n \vdash E_2}{(e_m e_n)_i \vdash (X_i \supseteq \boxed{\dots}) \wedge E_1 \wedge E_2}
 \end{array}$$

- 방정식 풀기: 모인 연립 방정식

$$(X_1 \supseteq \text{setexp}_1) \wedge (X_2 \supseteq \text{setexp}_2) \wedge \dots$$

을 풀면 된다. 연립 방정식을 푸는 방법은 방정식들($X_i \supseteq \text{setexp}_i$) 집합에서 부터 시작해서 새롭게 알게되는 방정식들을 계속 추가하는 것이다. 더 이상 추가할 게 없을 때 까지. 어떻게 추가하면 될까? 아래 규칙에 따라 추가한다.

$$\begin{array}{c}
 \frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq \{\lambda x.e\}}{\mathcal{X} \supseteq \{\lambda x.e\}} \qquad \frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq \emptyset}{\mathcal{X} \supseteq \emptyset} \\
 \\
 \frac{\mathcal{X} \supseteq \mathcal{Y} \cup \mathcal{Z}}{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{X} \supseteq \mathcal{Z}} \qquad \frac{\mathcal{X} \supseteq \mathcal{Y} @ \mathcal{Z} \quad \mathcal{Y} \supseteq \{\lambda x.e_i\}}{X_x \supseteq \mathcal{Z} \quad \mathcal{X} \supseteq X_i}
 \end{array}$$

첫 세개의 규칙은 알게 된 사실을 단순히 전파하거나 잘게 녹이는 규칙이다.

네번째 규칙은 함수호출식의 경우 알게 된 사실을 전파하는 것이다. \mathcal{X} 가 \mathcal{Y} -호출- \mathcal{Z} 의 결과를 가지게 되고 ($\mathcal{X} \supseteq \mathcal{Y} @ \mathcal{Z}$), \mathcal{Y} 가 어떤 함수인지를 알게되면 ($\mathcal{Y} \supseteq \{\lambda x.e_i\}$), 함수 인자 X_x 는 \mathcal{Z} 을 가지게 되고 ($X_x \supseteq \mathcal{Z}$) 호출식 결과 \mathcal{X} 는 그 함수의 몸통식 e_i 의 결과 X_i 를 가지게 된다 ($\mathcal{X} \supseteq X_i$).

- 방정식의 해: 위의 방식으로 모두 모아가면 언젠가는 끝난다. 프로그램을 구성하는 식의 갯수와 함수식의 갯수는 유한하기 때문이다. 방정식 변수 X_i 의 답은 위와같이 모은 방정식중에서 다음과 같은 명백한 꼴들을 모두 모으면 된다.

$$X_i \supseteq \{\lambda x.e\} \quad X_i \supseteq \{\lambda y.e'\} \quad \dots$$

위에서 오른쪽에 있는 함수식들을 모은 집합

$$\{\lambda x.e, \lambda y.e', \dots\},$$

이 집합이 식 e_i 가 실행중에 가지는 함수식들을 모두 포함하게 된다. □