Homework 8

SNU 4190.310, 2022 봄

Kwangkeun Yi

Due: 6/10(금), 24:00

이번 숙제의 목적은:

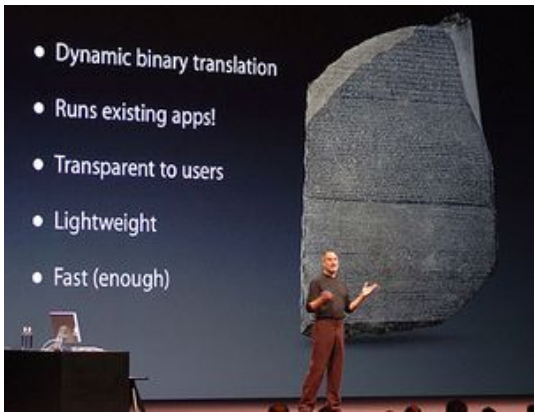
- 애플사에서 자랑하는 하위언어 자동번역 기술을 손수 이뤄보기. (특별한 기술이 아님을 겪고 자신감을 가지기)
- M언어로 짜여진 프로그램이 무난히 실행될 수 있는지를 미리 확인해 주는 정교한 안전장치를 갖추어 보기. 즉, let-다형 타입 시스템(let-polymorphic type system)을 장착해서 편리하면서도 안전한 프로그래밍 환경을 완성해 보기.

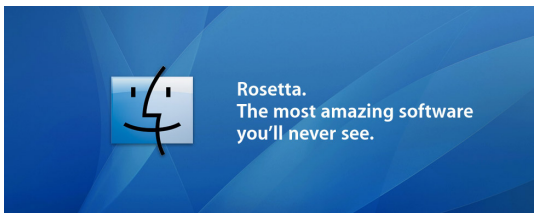
Exercise 1 (40pts) "SM5 RosettaX"

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 바꿔탈 때 개발한 Rosetta 소프트웨어의 새 버전

이다.





Rosetta 소프트웨어의 엔진은 애플이 "Transitive"라는 회사

에서 구입한 기계어 번역기였다. 당시 Transitive사는 영국 맨체스터에 본사를 둔 아주 작은 벤쳐회사 였다.[1]

이번 숙제는 이러한 번역기 RosettaX를 SM5에 대해서 만들어보는 것이다. 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 K부품을 건드렸던 `call` 명령의 의미와 빈명령문의 의미는

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

Sonata에서는 다음과 같다:

$$\begin{array}{rllll} & (l :: v :: (x, C', E') :: S, & M, & E, & \texttt{call} :: C) \\ \Rightarrow & (S, & M\{l \mapsto v\}, & (x, l) :: E', & C') \end{array}$$

$$\begin{array}{rllll} & (S, & M, & E, & \mathit{empty}) \\ \Rightarrow & (S, & M, & E, & \mathit{empty}) \end{array}$$

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

$$\begin{array}{rcrcl} S & \in & \mathit{Stack} & = & \mathit{Svalue\ list} \\ M & \in & \mathit{Memory} & = & \mathit{Loc} \to \mathit{Svalue} \end{array}$$

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 2 (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
```

□