

Homework 6  
SNU 4190.310, 2023 봄  
Kwangkeun Yi  
**Due: 5/15(월) 24:00**

이번 숙제의 목적은:

- 상위언어의 실행을 하위언어로 실현할 때 드러나는 프로그램 실행에 필요한 부품/개념들을 체험해보기.
- 언어사이의 자동 번역기를 제작해보기.
- 메모리 재활용의 기본개념을 상위에서 구현해보기.
- 람다 계산법(Lambda Calculus) 실행기를 제작하고 람다식으로 놀아보기.

**Exercise 1** (40pts) “SM5”

K--(교재 4.3) 프로그램들을 가상 기계(abstract machine)인 SM5에서 실행될 수 있도록 번역하는 번역기를 제작한다.

SM5는 가상의 기계이다. “SM”은 “Stack Machine”을 뜻하고, “5”는 그 기계의 부품이 5개이기 때문이다:

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

$S$ 는 스택,  $M$ 은 메모리,  $E$ 는 환경,  $C$ 는 명령어,  $K$ 는 남은 할 일(“continuation”이라고 부름)을 뜻하고 다음 집합들의 원소이다:

$$\begin{aligned}
S &\in \text{Stack} = \text{Svalue list} \\
M &\in \text{Memory} = \text{Loc} \rightarrow \text{Value} \\
E &\in \text{Environment} = (\text{Var} \times (\text{Loc} + \text{Proc})) \text{ list} \\
C &\in \text{Command} = \text{Cmd list} \\
K &\in \text{Continuation} = (\text{Command} \times \text{Environment}) \text{ list} \\
\\
v &\in \text{Value} = \text{Integer} + \text{Bool} + \{\cdot\} + \text{Record} + \text{Loc} \\
x &\in \text{Var} \\
\langle a, o \rangle, l &\in \text{Loc} = \text{Base} \times \text{Offset} \\
&\text{Offset} = \text{Integer} \\
z &\in \text{Integer} \\
b &\in \text{Bool} \\
r &\in \text{Record} = (\text{Var} \times \text{Loc}) \text{ list} \\
w &\in \text{Svalue} = \text{Value} + \text{Proc} + (\text{Var} \times \text{Loc}) + (\text{Var} \times \text{Proc}) \quad (* \text{ stackable values } *) \\
p &\in \text{Proc} = \text{Var} \times \text{Command} \times \text{Environment} \\
&\text{Cmd} = \{\text{push } v, \text{push } x, \text{push}(x, C), \\
&\quad \text{pop}, \text{store}, \text{load}, \text{jtr}(C, C), \\
&\quad \text{malloc}, \text{box } z, \text{unbox } x, \text{bind } x, \text{unbind}, \text{get}, \text{put}, \text{call}, \\
&\quad \text{add}, \text{sub}, \text{mul}, \text{div}, \text{eq}, \text{less}, \text{not}\}
\end{aligned}$$

기계의 작동은 다음과 같이 기계의 상태가 변화하는 과정으로 정의할 수 있다:

$$(S, M, E, C, K) \Rightarrow (S', M', E', C', K')$$

언제 어떻게 위의 기계작동의 한 스텝( $\Rightarrow$ )이 일어나는 지는 다음과 같다:

$$\begin{array}{l} (S, \quad \quad \quad M, \ E, \quad \text{push } v :: C, \ K) \\ \Rightarrow (v :: S, \quad \quad \quad M, \ E, \quad \quad \quad C, \ K) \end{array}$$

$$\begin{array}{l} (S, \quad \quad \quad M, \ E, \quad \text{push } x :: C, \ K) \\ \Rightarrow (w :: S, \quad \quad \quad M, \ E, \quad \quad \quad C, \ K) \text{ if } (x, w) \text{ is the first such entry in } E \end{array}$$

$$\begin{array}{l} (S, \quad \quad \quad M, \ E, \quad \text{push } (x, C') :: C, \ K) \\ \Rightarrow ((x, C', E) :: S, \quad \quad \quad M, \ E, \quad \quad \quad C, \ K) \end{array}$$

$$\begin{array}{l} (w :: S, \quad \quad \quad M, \ E, \quad \quad \quad \text{pop} :: C, \ K) \\ \Rightarrow (S, \quad \quad \quad M, \ E, \quad \quad \quad C, \ K) \end{array}$$

$$\begin{array}{l} (l :: v :: S, \quad \quad \quad M, \ E, \quad \quad \quad \text{store} :: C, \ K) \\ \Rightarrow (S, \quad \quad \quad M\{l \mapsto v\}, \ E, \quad \quad \quad C, \ K) \end{array}$$

$$\begin{array}{l} (l :: S, \quad \quad \quad M, \ E, \quad \quad \quad \text{load} :: C, \ K) \\ \Rightarrow (M(l) :: S, \quad \quad \quad M, \ E, \quad \quad \quad C, \ K) \end{array}$$

$(true :: S,$	$M,$	$E,$	$\text{jtr}(C_1, C_2) :: C, K)$
$\Rightarrow (S,$	$M,$	$E,$	$C_1 :: C, K)$
$(false :: S,$	$M,$	$E,$	$\text{jtr}(C_1, C_2) :: C, K)$
$\Rightarrow (S,$	$M,$	$E,$	$C_2 :: C, K)$
$(S,$	$M,$	$E,$	$\text{malloc} :: C, K)$
$\Rightarrow (\langle a, 0 \rangle :: S,$	$M,$	$E,$	$C, K) \text{ new } a$
$(w_1 :: \dots :: w_z :: S,$	$M,$	$E,$	$\text{box } z :: C, K)$
$\Rightarrow ([w_1, \dots, w_z] :: S,$	$M,$	$E,$	$C, K)$
$([w_1, \dots, w_z] :: S,$	$M,$	$E,$	$\text{unbox } x :: C, K)$
$\Rightarrow (v :: S,$	$M,$	$E,$	$C, K) \ w_k = (x, v), 1 \leq k \leq z$
$(w :: S,$	$M,$	$E,$	$\text{bind } x :: C, K)$
$\Rightarrow (S,$	$M,$	$(x, w) :: E,$	$C, K)$
$(S,$	$M,$	$(x, w) :: E,$	$\text{unbind} :: C, K)$
$\Rightarrow ((x, w) :: S,$	$M,$	$E,$	$C, K)$
$(l :: v :: (x, C', E') :: S,$	$M,$	$E,$	$\text{call} :: C, K)$
$\Rightarrow (S,$	$M\{l \mapsto v\},$	$(x, l) :: E',$	$C', (C, E) :: K)$
$(S,$	$M,$	$E,$	$\text{empty}, (C, E') :: K)$
$\Rightarrow (S,$	$M,$	$E',$	$C, K)$
$(S,$	$M,$	$E,$	$\text{get} :: C, K)$
$\Rightarrow (z :: S,$	$M,$	$E,$	$C, K) \text{ read } z \text{ from outside}$
$(z :: S,$	$M,$	$E,$	$\text{put} :: C, K)$
$\Rightarrow (S,$	$M,$	$E,$	$C, K) \text{ print } z \text{ and newline}$

$$\begin{aligned}
& (v_2 :: v_1 :: S, \quad M, \quad E, \quad \mathbf{add} :: C, \quad K) \\
\Rightarrow & (\mathbf{plus}(v_1, v_2) :: S, \quad M, \quad E, \quad C, \quad K) \\
& (v_2 :: v_1 :: S, \quad M, \quad E, \quad \mathbf{sub} :: C, \quad K) \\
\Rightarrow & (\mathbf{minus}(v_1, v_2) :: S, \quad M, \quad E, \quad C, \quad K) \\
& (z_2 :: z_1 :: S, \quad M, \quad E, \quad \mathbf{mul} :: C, \quad K) \\
\Rightarrow & ((z_1 * z_2) :: S, \quad M, \quad E, \quad C, \quad K) \quad \text{similar for } \mathbf{div} \\
& (v_2 :: v_1 :: S, \quad M, \quad E, \quad \mathbf{eq} :: C, \quad K) \\
\Rightarrow & (\mathbf{equal}(v_1, v_2) :: S, \quad M, \quad E, \quad C, \quad K) \\
& (v_2 :: v_1 :: S, \quad M, \quad E, \quad \mathbf{less} :: C, \quad K) \\
\Rightarrow & (\mathbf{less}(v_1, v_2) :: S, \quad M, \quad E, \quad C, \quad K) \\
& (b :: S, \quad M, \quad E, \quad \mathbf{not} :: C, \quad K) \\
\Rightarrow & (\mathbf{-}b :: S, \quad M, \quad E, \quad C, \quad K)
\end{aligned}$$

$$\begin{aligned}
\mathbf{less}(z_1, z_2) &= z_1 < z_2 \\
\mathbf{plus}(z_1, z_2) &= z_1 + z_2 \\
\mathbf{plus}(\langle a, z_1 \rangle, z_2) &= \langle a, z_1 + z_2 \rangle \quad \text{if } z_1 + z_2 \geq 0 \\
\mathbf{plus}(z_1, \langle a, z_2 \rangle) &= \langle a, z_1 + z_2 \rangle \quad \text{if } z_1 + z_2 \geq 0 \\
\mathbf{minus}(z_1, z_2) &= z_1 - z_2 \\
\mathbf{minus}(\langle a, z_1 \rangle, z_2) &= \langle a, z_1 - z_2 \rangle \quad \text{if } z_1 - z_2 \geq 0 \\
\mathbf{equal}(z_1, z_2) &= z_1 = z_2 \\
\mathbf{equal}(b_1, b_2) &= b_1 = b_2 \\
\mathbf{equal}(\cdot, \cdot) &= \mathbf{true} \\
\mathbf{equal}(r_1, r_2) &= (\forall \langle x, l \rangle \in r_1 : \langle x, l \rangle \in r_2) \wedge (\forall \langle x, l \rangle \in r_2 : \langle x, l \rangle \in r_1) \\
\mathbf{equal}(\langle a_1, z_1 \rangle, \langle a_2, z_2 \rangle) &= a_1 = a_2 \wedge z_1 = z_2 \\
\mathbf{equal}(-, -) &= \mathbf{false}
\end{aligned}$$

SM5의 프로그램  $C$ 를 실행한다는 것은,  $C$ 만 가지고 있는 빈 기계상태를 위에서 정의한 방식으로 변환해 간다는 뜻이다:

$$(\mathbf{empty}, \mathbf{empty}, \mathbf{empty}, C, \mathbf{empty}) \Rightarrow \dots \Rightarrow \dots$$

예를들어,

```
push 1 :: push 2 :: add :: put :: empty
```

는 K-- 프로그램 `write 1+2`과 같은 일을 하게 된다.

여러분이 할 것은, 잘 돌아가는 K-- 프로그램을 입력으로 받아서 같은 일을 하는 SM5 프로그램으로 변환하는 함수

```
trans: K.program -> Sm5.command
```

를 작성하는 것이다.

`trans`가 제대로 정의되었는지는, K-- 프로그램  $E$ 에 대해서,  $K.run(E)$ 와  $Sm5.run(trans(E))$ 을 실행해서 확인할 수 있을 것이다.

모듈 `Sm5`, 모듈 `K`, 그리고 K--의 파서는 제공된다(TA 페이지 참고). □

### Exercise 2 (20pts) “SM5 Limited = SM5 + 메모리 재활용”

SM5 메모리에서는 무한히 많은 새로운 주소가 샘솟을 수 없다.

이제, SM5의 메모리는 128개의 주소만 있다고 하자. 위의 문제에서 주어진 모듈 `Sm5`를 뜯어 고쳐서, `malloc`할 것이 더이상 없을 때 메모리를 재활용하는 함수 `gc`를 장착하라. 즉,

$$(S, M, E, \text{malloc} :: C, K) \Rightarrow (l :: S, M, E, C, K) \quad \text{new } l$$

이 아래와 같이 변경될 것이다:

$$(S, M, E, \text{malloc} :: C, K) \Rightarrow (l :: S, M, E, C, K) \quad \text{new } l, \text{ if } |\text{dom}M| < 128$$
$$(S, M, E, \text{malloc} :: C, K) \Rightarrow (l :: S, \text{gc}(\dots), E, C, K) \quad \text{recycled } l, \text{ if } |\text{dom}M| = 128$$

재활용함수 `gc`는 실제 구현보다 훨씬 간단하다. 현재 메모리에서 미래에 사용할 수 있는 부분만을 모으면 될 것이다. 그러한 부분들은 현재 기계 상태의 세 개의 부품에서 부터 도달 가능한 모든 메모리 주소들이 될 것이다.

□

### Exercise 3 (40pts) “Lambda Ground”

람다 계산법에서 “normal-order reduction” 룰을 따르는 실행기

```
reduce: lexp -> lexp
```

를 구현하라. 타입 `lexp`는 다음과 같다:

```
type lexp = Var of string
           | Lam of string * lexp
           | App of lexp * lexp
```

구현한 실행기를 가지고 람다식으로 자연수, 참, 거짓, 덧셈, 곱셈, 조건문, 재귀 함수등을 정의해보고 구현한 실행기로 돌려보며 결과를 확인해 보자. TA는 이런 경우등을 테스트해 볼 예정이다.

TA는 바뀐 연산자 `subst`와 람다식 파서를 제공할 것이다.

```
type substitution = (string * lexp) list
subst: substitution * lexp -> lexp
```

람다식의 스트링 문법(concrete syntax)는 다음과 같다:

$E ::=$	$var$	single-char lower-or-upper alphabet followed by zero or more apostrophes
	$\backslash var . E$	
	$EE$	left-associative and highest-precedence in parsing
	$( E )$	
	$let\ var = E\ in\ E$	

예를들어 람다식을 다음과 같이 쓰면 된다:

```
let a'' = \s.\z.s(sz) in
let Y = \f.(\x.f(xx))(\x.f(xx)) in
Y(\f.\n.n(fa''))
```

□