Homework 6

SNU 4190.310, 2025 봄

Kwangkeun Yi

Due: 5/16(Fri) 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-*[1] 프로그램들을 가상기계(abstract machine)어로 번역하는 번역기를 제작한다.

가상기계어의 의미(semantics)는 가상기계 SM5가 그 기계어를 어떻게 실행하는 지를 보면 알 수 있다. "SM5"의 "SM"은 "Stack Machine"을 뜻하고, "5"는 그 기계의 부품이 5개이기 때문이다:

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

S는 스택, M은 메모리, E는 환경, C는 명령어, K는 마저할일("continuation"이라고 부름)을 뜻하고 다음 집합들의 원소이다:

[1] 숙제4의 K-와 숙제3의 K-- 사이의 언어. 정확한 정의는 TA 페이지 참고.

$$
\begin{array}{rcrcl}
S & \in & \mathit{Stack} & = & \mathit{Svalue\ list} \\
M & \in & \mathit{Memory} & = & \mathit{Loc} \to \mathit{Value} \\
E & \in & \mathit{Environment} & = & (\mathit{Var} \times (\mathit{Loc} + \mathit{Proc}))\ \mathit{list} \\
C & \in & \mathit{Command} & = & \mathit{Cmd\ list} \\
K & \in & \mathit{Continuation} & = & (\mathit{Command} \times \mathit{Environment})\ \mathit{list}
\end{array}
$$

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

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

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

기계 작동의 한 스텝(⇒)의 정의는 다음과 같다:

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

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

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

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

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

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

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

$$
\begin{array}{rlllll}
 & (v_2 :: v_1 :: S, & M, & E, & \texttt{add} :: C, & K) \\
\Rightarrow & (plus(v_1, v_2) :: S, & M, & E, & C, & K)
\end{array}
$$

$$
\begin{array}{rlllll}
 & (v_2 :: v_1 :: S, & M, & E, & \texttt{sub} :: C, & K) \\
\Rightarrow & (minus(v_1, v_2) :: S, & M, & E, & C, & K)
\end{array}
$$

$$
\begin{array}{rlllll}
 & (z_2 :: z_1 :: S, & M, & E, & \texttt{mul} :: C, & K) \\
\Rightarrow & ((z_1 * z_2) :: S, & M, & E, & C, & K)
\end{array}
$$

similar for `div`

$$
\begin{array}{rlllll}
 & (v_2 :: v_1 :: S, & M, & E, & \texttt{eq} :: C, & K) \\
\Rightarrow & (equal(v_1, v_2) :: S, & M, & E, & C, & K)
\end{array}
$$

$$
\begin{array}{rlllll}
 & (v_2 :: v_1 :: S, & M, & E, & \texttt{less} :: C, & K) \\
\Rightarrow & (less(v_1, v_2) :: S, & M, & E, & C, & K)
\end{array}
$$

$$
\begin{array}{rlllll}
 & (b :: S, & M, & E, & \texttt{not} :: C, & K) \\
\Rightarrow & (\neg b :: S, & M, & E, & C, & K)
\end{array}
$$

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

SM5의 프로그램 C의 실행은(C의 의미는), C만 가지고 있는 빈 기계상태를 위의 정의대로 한 스텝 한 스텝 변환해 가는 과정이다:

$$(empty, empty, empty, C, empty) \Rightarrow \cdots \Rightarrow \cdots$$

예를들어,

$$\texttt{push 1} :: \texttt{push 2} :: \texttt{add} :: \texttt{put} :: \mathit{empty}$$

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

$$\texttt{trans: K.program} \rightarrow \texttt{Sm5.command}$$

를 작성하는 것이다.

`trans`가 제대로 정의되었는지는, K-* 프로그램 E에 대해서, `K.run(`E`)`와 `Sm5.run(trans(`E`))`을 실행해서 테스트해 볼 수 있다.

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

Exercise 2 (30pts) "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 gc를 장착하라. 즉,

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

이 아래와 같이 변경될 것이다:

$$\begin{array}{lcll} (S, M, E, \texttt{malloc} :: C, K) & \Rightarrow & (l :: S, M, E, C, K) & \text{new } l, \text{ if } |\mathrm{dom}M| < 128 \\ (S, M, E, \texttt{malloc} :: C, K) & \Rightarrow & (l :: S, \mathrm{gc}(\cdots), E, C, K) & \text{recycled } l, \text{ if } |\mathrm{dom}M| = 128 \end{array}$$

재활용함수 gc는 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

$$\texttt{reduce: lexp} \rightarrow \texttt{lexp}$$

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, 재귀함수등을 정의해보고 구현한 실행기로 돌려보며 결과를 확인해 보자. TA는 이런 경우등을 테스트해 볼 예정이다.

TA는 바꿔치기 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
	|	`\` *var* `.` E	
	|	$E\,E$	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''))
```

□