The M Language

SNU4190.310

1 Syntax

The syntax of M is:

e	$::=$	*const*	constant
	$\mid$	*id*	identifier
	$\mid$	`fn` *id* `=>` *e*	function
	$\mid$	*e e*	application
	$\mid$	`let` *bind* `in` *e* `end`	local block
	$\mid$	`if` *e* `then` *e* `else` *e*	branch
	$\mid$	*e op e*	infix binary operation
	$\mid$	`read`	input
	$\mid$	`write` *e*	output
	$\mid$	`(`*e*`)`	
	$\mid$	`malloc` *e*	allocation
	$\mid$	*e* `:=` *e*	assignment
	$\mid$	`!`*e*	bang, dereference
	$\mid$	*e* `;` *e*	sequence
	$\mid$	`(`*e*`,`*e*`)`	pair
	$\mid$	*e*`.1`	first component
	$\mid$	*e*`.2`	second component
bind	$::=$	`val` *id* `=` *e*	value binding
	$\mid$	`rec` *id* `=` `fn` *id* `=>` *e*	recursive function binding
op	$::=$	`+` $\mid$ `-` $\mid$ `=` $\mid$ `and` $\mid$ `or`	
const	$::=$	`true` $\mid$ `false` $\mid$ *string* $\mid$ *num*	
id			alpha-numeric identifier
string			string
num			integer

1.1 Program

A program is an expression of non-function type: i, s, b, τ loc, or $\tau \times \tau'$ where τ and τ' are non-function types. For example,

```
fn x => x
```

is not a program, because its type is a function. On the other hand,

```
(fn x => x) read
```

is a program, whose type is integer.

1.2 Identifiers

Alpha-numeric identifiers are [a-zA-Z][a-zA-Z0-9_']*. Identifiers are case sensitive: `z` and `Z` are different. The reserved words cannot be used as identifiers: `fn let in end if then else read write malloc val rec and or true false`

1.3 Strings/Numbers/Comments

Strings begin and end with `"`. Inside the two double quotes, any sequence of characters excepting the new-line and the `"` characters can appear: [^"^\n]*. Null string `""` is possible.

Numbers are integers, optionally prefixed with `~`(for negative integer): `~`?[0-9]+.

A comment is any character sequence within the comment block `(* *)`. The comment block can be nested.

1.4 Precedence/Associativity

In parsing M program text, the precedence of the M constructs in decreasing order is as follows. Symbols in the same set have identical precedence. Symbols with subscript L (respectively R) are left (respectively right) associative.

$$
\begin{aligned}
&\{\text{function application}\}_L,\\
&\{\texttt{.1}\}_L, \{\texttt{.2}\}_L,\\
&\{\texttt{!}, \texttt{malloc}\}_R,\\
&\{\texttt{and}\}_L,\\
&\{\texttt{+}, \texttt{-}, \texttt{or}\}_L,\\
&\{\texttt{=}\}_L,\\
&\{\texttt{if}\}_R,\\
&\{\texttt{:=}\}_R,\\
&\{\texttt{write}\}_R,\\
&\{\texttt{fn}\}_R,\\
&\{\texttt{;}\}_L
\end{aligned}
$$

For example, M program

```
fn x => x := y := 1; !x!x
```

is parsed as

```
(fn x => x := (y := 1) ; !(x (!x))
```

not as

```
(fn x => x) := (y := 1); ((!x) (!x))
```

nor as

```
fn x => (x := (y := 1; ((!x) (!x))))
```

Rule of thumb: for your test programs, if your programs are hard to read (hence can be parsed not as you expected) then put parentheses around.

2 Dynamic Semantics

$$
\begin{array}{rcllr}
x, f & \in & Id & & \text{identifiers} \\
v & \in & Val & = Num + String + Bool + Loc + Pair + Closure & \text{values} \\
n & \in & Num & & \\
s & \in & String & & \\
b & \in & Bool & & \\
l & \in & Loc & & \\
\langle v_1, v_2 \rangle & \in & Pair & = Val \times Val & \text{pair values} \\
 & & Closure & = Fexpr \times Env & \text{function values} \\
\sigma & \in & Env & = Id \xrightarrow{\text{fin}} Val & \text{environments} \\
 & & Fexpr & = \lambda x.e & \text{function defs} \\
 & & & \mid\ f\lambda x.e & \text{rec function defs} \\
M & \in & Memory & = Loc \xrightarrow{\text{fin}} Val & \text{memories}
\end{array}
$$

Notation:

- We write $\{x_1 \mapsto v_1, \cdots, x_n \mapsto v_n\}$ for a finite function f. The domain $Dom(f)$ is $\{x_1, \cdots, x_n\}$.
- We write $f(x)$ for v if $x \mapsto v \in f$. If $x \mapsto v \notin f$ then $f(x)$ is not defined.
- We write $f[x \mapsto v]$ for

$$
\begin{array}{rl}
f \cup \{x \mapsto v\} & \text{if } x \notin Dom(f) \\
(f \setminus \{x \mapsto f(x)\}) \cup \{x \mapsto v\} & \text{if } x \in Dom(f).
\end{array}
$$

The semantics rules precisely defines how relations of the form

$$\sigma, M \vdash e \Rightarrow v, M'$$

to be inferred. The relation is read "expression e computes value v under environment σ and memory M."

Definition 1 (Program's Semantics) *A program e's semantics is defined to be the inference tree of relation $\emptyset, \emptyset \vdash e \Rightarrow v, M$ for some v and M. If there is no such v and M, then the expression has no meaning.*

[Const] $$\sigma, M \vdash \ \mathit{const} \ \Rightarrow \ \mathit{const} \text{ in } \mathit{Val}, M$$

[Id] $$\frac{\sigma(x) = v}{\sigma, M \vdash \ x \ \Rightarrow \ v, M}$$

[Fun] $$\sigma, M \vdash \ \texttt{fn} \ x \ \texttt{=>} \ e \ \Rightarrow \ \langle \lambda x.e, \sigma \rangle, M$$

[App] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ \langle \lambda x.e, \sigma' \rangle, M' \qquad \sigma, M' \vdash \ e_2 \ \Rightarrow \ v_2, M'' \qquad \sigma'[x \mapsto v_2], M'' \vdash \ e \ \Rightarrow \ v, M'''}{\sigma, M \vdash \ e_1 \ e_2 \ \Rightarrow \ v, M'''}$$

[RecApp] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ \langle f \lambda x.e, \sigma' \rangle, M' \qquad \sigma, M' \vdash \ e_2 \ \Rightarrow \ v_2, M'' \qquad \sigma'[x \mapsto v_2][f \mapsto \langle f \lambda x.e, \sigma' \rangle], M'' \vdash \ e \ \Rightarrow \ v, M'''}{\sigma, M \vdash \ e_1 \ e_2 \ \Rightarrow \ v, M'''}$$

[Let] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ v_1, M' \qquad \sigma[x \mapsto v_1], M' \vdash \ e_2 \ \Rightarrow \ v, M''}{\sigma, M \vdash \ \texttt{let} \ x \ \texttt{=} \ e_1 \ \texttt{in} \ e_2 \ \texttt{end} \Rightarrow v, M''}$$

[RecLet] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ \langle \lambda x.e, \sigma' \rangle, M' \qquad \sigma[f \mapsto \langle f \lambda x.e, \sigma' \rangle], M' \vdash \ e_2 \ \Rightarrow \ v, M''}{\sigma, M \vdash \ \texttt{let rec} \ f \ \texttt{=} \ e_1 \ \texttt{in} \ e_2 \ \texttt{end} \Rightarrow v, M''}$$

[IfTrue] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ \text{true}, M' \qquad \sigma, M' \vdash \ e_2 \ \Rightarrow \ v, M''}{\sigma, M \vdash \ \texttt{if} \ e_1 \ \texttt{then} \ e_2 \ \texttt{else} \ e_3 \ \Rightarrow \ v, M''}$$

[IfFalse] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ \text{false}, M' \qquad \sigma, M' \vdash \ e_3 \ \Rightarrow \ v, M''}{\sigma, M \vdash \ \texttt{if} \ e_1 \ \texttt{then} \ e_2 \ \texttt{else} \ e_3 \ \Rightarrow \ v, M''}$$

[Op] $$\frac{\sigma, M \vdash \ e_1 \ \Rightarrow \ v_1, M' \qquad \sigma, M' \vdash \ e_2 \ \Rightarrow \ v_2, M'' \qquad op(v_1, v_2) = v}{\sigma, M \vdash \ e_1 \ op \ e_2 \ \Rightarrow \ v, M''}$$

[Read] $$\sigma, M \vdash \ \texttt{read} \ \Rightarrow \ n, M$$

[Write] $$\frac{\sigma, M \vdash \ e \ \Rightarrow \ v, M' \qquad v \in \mathit{Num} + \mathit{String} + \mathit{Bool}}{\sigma, M \vdash \ \texttt{write} \ e \ \Rightarrow \ v, M'}$$

[Paren] $$\frac{\sigma, M \vdash \ e \ \Rightarrow \ v, M'}{\sigma, M \vdash \ \texttt{(}e\texttt{)} \ \Rightarrow \ v, M'}$$

[Malloc] $$\frac{\sigma, M \vdash e \Rightarrow v, M' \qquad l \notin Dom(M')}{\sigma, M \vdash \texttt{malloc}\ e \Rightarrow l, M'[l \mapsto v]}$$

[Assign] $$\frac{\sigma, M \vdash e_1 \Rightarrow l, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash e_1 \texttt{ := } e_2 \Rightarrow v, M''[l \mapsto v]}$$

[Bang] $$\frac{\sigma, M \vdash e \Rightarrow l, M' \qquad v = M'(l)}{\sigma, M \vdash \texttt{!}e \Rightarrow v, M'}$$

[Seq] $$\frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash e_1 \texttt{ ; } e_2 \Rightarrow v_2, M_2}$$

[Pair] $$\frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash \texttt{(}e_1\texttt{,}e_2\texttt{)} \Rightarrow \langle v_1, v_2 \rangle, M_2}$$

[Compo1] $$\frac{\sigma, M \vdash e \Rightarrow \langle v_1, v_2 \rangle, M'}{\sigma, M \vdash e\texttt{.1} \Rightarrow v_1, M'}$$

[Compo2] $$\frac{\sigma, M \vdash e \Rightarrow \langle v_1, v_2 \rangle, M'}{\sigma, M \vdash e\texttt{.2} \Rightarrow v_2, M'}$$

$$\frac{}{\texttt{+}(n_1, n_2) = n_1 + n_2} \qquad \frac{}{\texttt{-}(n_1, n_2) = n_1 - n_2}$$

$$\frac{}{\texttt{and}(b_1, b_2) = b_1 \wedge b_2} \qquad \frac{}{\texttt{or}(b_1, b_2) = b_1 \vee b_2}$$

$$\frac{}{\texttt{=}(n, n) = \text{true}} \qquad \frac{}{\texttt{=}(s, s) = \text{true}} \qquad \frac{}{\texttt{=}(b, b) = \text{true}} \qquad \frac{}{\texttt{=}(l, l) = \text{true}}$$

$$\frac{n_1 \neq n_2}{\texttt{=}(n_1, n_2) = \text{false}} \qquad \frac{s_1 \neq s_2}{\texttt{=}(s_1, s_2) = \text{false}} \qquad \frac{b_1 \neq b_2}{\texttt{=}(b_1, b_2) = \text{false}} \qquad \frac{l_1 \neq l_2}{\texttt{=}(l_1, l_2) = \text{false}}$$

3 Static Semantics: Type System

Type			
τ	$::=$	i	integer type
	$\mid$	b	boolean type
	$\mid$	s	string type
	$\mid$	$\tau \times \tau$	pair type
	$\mid$	τ loc	location type
	$\mid$	$\tau \to \tau$	function type

타입규칙은 다음의 관계를 결정해주는 규칙들이다:

$$\Gamma \vdash e : \tau$$

위의 관계를 다음과 같이 읽자 "프로그램 식 e는 Γ라는 환경에서 타입 τ를 가진다." 타입 환경 Γ는 다음과 같은 테이블이다:

$$\Gamma \in \mathit{TypeEnv} = \mathit{Id} \xrightarrow{\text{fin}} \mathit{Type} \qquad \text{type environment}$$

Definition 2 (Program's Type) *A program e has type τ iff relation $\emptyset \vdash e : \tau$ is proved. If there is no such τ, then the expression has no type.*

다음이 프로그램 식 e 의 타입을 결정하는 규칙들이다. 완성해서 사용하라:

[Num] $$\Gamma \vdash n : i$$

[Bool] $$\Gamma \vdash \texttt{true} : b \qquad \Gamma \vdash \texttt{false} : b$$

[String] $$\Gamma \vdash \mathit{string} : s$$

[Fun] $$\frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \texttt{fn}\ x\ \texttt{=>}\ e : \tau_1 \to \tau_2}$$

[App] $$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

[Let] $$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} : \tau_2}$$

[RecLet] $$\frac{\vdash : \qquad \vdash :}{\Gamma \vdash \texttt{let rec}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} : \tau_2}$$

[If] $$\frac{\Gamma \vdash e_1 : b \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

[Read] $$\Gamma \vdash \texttt{read} : i$$

[Write] $$\frac{\Gamma \vdash e : \tau \quad \tau = i, b, \text{or } s}{\Gamma \vdash \texttt{write}\ e : \tau}$$

[Malloc] $$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{malloc}\ e : \tau\ \mathrm{loc}}$$

[Assign] $$\frac{\Gamma \vdash e_1 : \tau\ \mathrm{loc} \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ \texttt{:=}\ e_2 : \tau}$$

[Bang] $$\frac{\vdash :}{\Gamma \vdash \texttt{!}e : \tau}$$

[Seq] $$\frac{\vdash : \qquad \vdash :}{\Gamma \vdash e_1\ \texttt{;}\ e_2 : \tau}$$

[Pair]

$$\frac{\vdash \; : \qquad \vdash \; :}{\Gamma \vdash \texttt{(}e_1\texttt{,}e_2\texttt{)} \; : \; \tau_1 \times \tau_2}$$

[Compo1]

$$\frac{\vdash \; :}{\Gamma \vdash e\texttt{.1} \; : \; \tau_1}$$

[Compo2]

$$\frac{\vdash \; :}{\Gamma \vdash e\texttt{.2} \; : \; \tau_2}$$

[Op]

$$\frac{\Gamma \vdash e_1 \; : \; i \qquad \Gamma \vdash e_2 \; : \; i}{\Gamma \vdash e_1 \, (\texttt{+/-}) \, e_2 \; : \; i}$$

[Op]

$$\frac{\Gamma \vdash e_1 \; : \; b \qquad \Gamma \vdash e_2 \; : \; b}{\Gamma \vdash e_1 \, (\texttt{and/or}) \, e_2 \; : \; b}$$

[Op]

$$\frac{\Gamma \vdash e_1 \; : \; \tau \qquad \Gamma \vdash e_2 \; : \; \tau \qquad \tau = i, b, s, \text{ or } l}{\Gamma \vdash e_1 \texttt{=} e_2 \; : \; b}$$