

The M Language

SNU 4541.574, Fall 2007

1 Syntax

The syntax of M is:

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

{function application} $_L$,
{.1} $_L$, {.2} $_L$,
{!, malloc} $_R$,
{and} $_L$,
{+, -, or} $_L$,
{=} $_L$,
{if} $_R$,
{:=} $_R$,
{write} $_R$,
{fn} $_R$,
{;} $_L$

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

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

Notation:

- We write $\{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$ for a finite function f . The domain $Dom(f)$ is $\{x_1, \dots, x_n\}$.
- 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

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

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]	$\sigma, M \vdash \text{const} \Rightarrow \text{const in Val}, M$
[Id]	$\frac{\sigma(x) = v}{\sigma, M \vdash x \Rightarrow v, M}$
[Fun]	$\sigma, M \vdash \text{fn } x \Rightarrow e \Rightarrow \langle \lambda x. e, \sigma \rangle, M$
[App]	$\frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x. e, \sigma' \rangle, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \quad \sigma'[x \mapsto v_2], M'' \vdash e \Rightarrow v, M'''}{\sigma, M \vdash e_1 e_2 \Rightarrow v, M'''}$
[RecApp]	$\frac{\sigma, M \vdash e_1 \Rightarrow \langle f \lambda x. e, \sigma' \rangle, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \quad \sigma'[x \mapsto v_2][f \mapsto \langle f \lambda x. e, \sigma' \rangle], M'' \vdash e \Rightarrow v, M'''}{\sigma, M \vdash e_1 e_2 \Rightarrow v, M'''}$
[Let]	$\frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \quad \sigma[x \mapsto v_1], M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \text{let } x = e_1 \text{ in } e_2 \text{ end} \Rightarrow v, M''}$
[RecLet]	$\frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x. e, \sigma' \rangle, M' \quad \sigma[f \mapsto \langle f \lambda x. e, \sigma' \rangle], M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \text{let rec } f = e_1 \text{ in } e_2 \text{ end} \Rightarrow v, M''}$
[IfTrue]	$\frac{\sigma, M \vdash e_1 \Rightarrow \text{true}, M' \quad \sigma, M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v, M''}$
[IfFalse]	$\frac{\sigma, M \vdash e_1 \Rightarrow \text{false}, M' \quad \sigma, M' \vdash e_3 \Rightarrow v, M''}{\sigma, M \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v, M''}$
[Op]	$\frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \quad \text{op}(v_1, v_2) = v}{\sigma, M \vdash e_1 \text{ op } e_2 \Rightarrow v, M''}$
[Read]	$\sigma, M \vdash \text{read} \Rightarrow n, M$
[Write]	$\frac{\sigma, M \vdash e \Rightarrow v, M' \quad v \in \text{Num} + \text{String} + \text{Bool}}{\sigma, M \vdash \text{write } e \Rightarrow v, M'}$
[Paren]	$\frac{\sigma, M \vdash e \Rightarrow v, M'}{\sigma, M \vdash (e) \Rightarrow v, M'}$

$$\begin{array}{c}
\text{[Malloc]} \quad \frac{\sigma, M \vdash e \Rightarrow v, M' \quad l \notin \text{Dom}(M')}{\sigma, M \vdash \text{malloc } e \Rightarrow l, M'[l \mapsto v]} \\
\text{[Assign]} \quad \frac{\sigma, M \vdash e_1 \Rightarrow l, M' \quad \sigma, M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash e_1 := e_2 \Rightarrow v, M''[l \mapsto v]} \\
\text{[Bang]} \quad \frac{\sigma, M \vdash e \Rightarrow l, M' \quad v = M'(l)}{\sigma, M \vdash !e \Rightarrow v, M'} \\
\text{[Seq]} \quad \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash e_1 ; e_2 \Rightarrow v_2, M_2} \\
\text{[Pair]} \quad \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash (e_1, e_2) \Rightarrow \langle v_1, v_2 \rangle, M_2} \\
\text{[Compo1]} \quad \frac{\sigma, M \vdash e \Rightarrow \langle v_1, v_2 \rangle, M'}{\sigma, M \vdash e.1 \Rightarrow v_1, M'} \\
\text{[Compo2]} \quad \frac{\sigma, M \vdash e \Rightarrow \langle v_1, v_2 \rangle, M'}{\sigma, M \vdash e.2 \Rightarrow v_2, M'} \\
\frac{}{\overline{+(n_1, n_2) = n_1 + n_2}} \quad \frac{}{\overline{-(n_1, n_2) = n_1 - n_2}} \\
\frac{}{\overline{\text{and}(b_1, b_2) = b_1 \wedge b_2}} \quad \frac{}{\overline{\text{or}(b_1, b_2) = b_1 \vee b_2}} \\
\frac{}{\overline{=(n, n) = \text{true}}} \quad \frac{}{\overline{=(s, s) = \text{true}}} \quad \frac{}{\overline{=(b, b) = \text{true}}} \quad \frac{}{\overline{=(l, l) = \text{true}}} \\
\frac{n_1 \neq n_2}{\overline{=(n_1, n_2) = \text{false}}} \quad \frac{s_1 \neq s_2}{\overline{=(s_1, s_2) = \text{false}}} \quad \frac{b_1 \neq b_2}{\overline{=(b_1, b_2) = \text{false}}} \quad \frac{l_1 \neq l_2}{\overline{=(l_1, l_2) = \text{false}}}
\end{array}$$