The M Specification

Aedan Smith

February 6, 2019

Contents

1 Syntax

$$\begin{array}{lll}
\langle\mathit{character}\rangle & ::= & \mathbb{C} \supseteq \langle\mathit{special + newline + whitespace}\rangle \\[2mm]
\langle\mathit{special}\rangle & ::= & \{`;\text{'},`"\text{'},`(\text{'},`)\text{'}\} \\[2mm]
\langle\mathit{comment}\rangle & ::= & `;\text{'} \; \langle\mathit{character - newline}\rangle^* \; \langle\mathit{newline}\rangle \\[2mm]
\langle\mathit{symbol}\rangle & ::= & \langle\mathit{character - special - whitespace}\rangle^* \\
& & | \quad `"\text{'} \; (\langle\mathit{character - "}\rangle \; | \; `\backslash\text{'} \; \langle\mathit{character}\rangle)^* \; `"\text{'} \\[2mm]
\langle\mathit{expression}\rangle & ::= & \langle\mathit{symbol}\rangle \\
& & | \quad `(\text{'} \; \langle\mathit{expression}\rangle^* \; `)\text{'} \\[2mm]
\langle\mathit{program}\rangle & ::= & \langle\mathit{expression}\rangle^*
\end{array}$$

Figure 1: The M grammar in EBNF.

1.1 Characters

M can use any character set which encodes the four special characters, a backslash, a newline character, and a whitespace character. For consistency, all M code samples in this specification use the ASCII character set.

1.2 Specials

The four special characters are reserved for use in other productions. This is done to ensure that symbols such as `this()` do not include any special characters which are meant be used for other syntactic constructs. Note that the backslash is not included in the list of special characters as it is only used inside of a literal symbol.

1.3 Comments

Comments in M begin with a semicolon and last until the end of the line. They are ignored and discarded like whitespace and newlines, as they are only intended to be used for explaining code.

1.4 Symbols

Symbols in M have two forms, inline and literal. Inline symbols are strings terminated by whitespace or a special symbol, and should be used by default. Literal symbols are escapable strings surrounded by quotes, and should only be used when a symbol is impossible to represent using inline symbols (for example, the symbol `"()"`).

1.5 Expressions

Expressions in M have two forms, symbols and lists. Symbol expressions are identical to symbols defined in section 1.4. Lists expressions are simply lists of expressions surrounded by matching parentheses.

1.6 Programs

M programs are lists of expressions. It is unspecified how these expressions are stored, so they can be in memory, in a single file, in multiple files, on the internet, or any other method which is convenient.

3

2 Semantics

$$
\begin{array}{ll}
\text{(Context)} & \dfrac{a \Downarrow i}{\langle a, \Gamma \rangle \Downarrow \langle i, \Gamma \rangle} \\[2ex]
\text{(Macro)} & \dfrac{\langle a, \Gamma \rangle \Downarrow i}{\langle (\texttt{a e}), \Gamma \rangle \Downarrow i(e)} \\[2ex]
\text{(Function)} & \dfrac{}{(\textbf{fn}\ \texttt{x a}) \Downarrow \lambda x.a} \\[2ex]
\text{(Impure)} & \dfrac{\langle a, \Gamma \rangle \Downarrow i}{\langle (\textbf{impure}\ \texttt{a}), \Gamma \rangle \Downarrow \pi i}
\end{array}
$$

$$
\begin{array}{ll}
\dfrac{\langle a, \Gamma \rangle \Downarrow \lambda x.i \quad \langle b, \Gamma \rangle \Downarrow j \quad \langle i[x/j] \rangle \Downarrow v}{\langle (\textbf{ap}\ \texttt{a b}), \Gamma \rangle \Downarrow v} & \text{(Apply-Function)} \\[2ex]
\dfrac{\langle a, \Gamma \rangle \Downarrow \pi i \quad \langle b, \Gamma \rangle \Downarrow j \quad \langle j(i) \rangle \Downarrow \pi v}{\langle (\textbf{ap}\ \texttt{a b}), \Gamma \rangle \Downarrow \pi v} & \text{(Apply-Impure)} \\[2ex]
\dfrac{\langle a, \Gamma \rangle \Downarrow i}{\langle (\textbf{def}\ \texttt{x a}), \Gamma \rangle \Downarrow \langle i, \Gamma(x) = i \rangle} & \text{(Define)} \\[2ex]
\dfrac{}{\langle a, \Gamma \rangle \Downarrow \Gamma(a)} & \text{(Symbol)}
\end{array}
$$

Figure 2: The natural semantics of M.

2.1 Define

Def expressions are expressions of the form (**def** name value). They state that all symbols name evaluate to value, and evaluate to the the value they define. Multiple def expressions with the same name are invalid.

2.2 Symbol

Symbol expressions are expressions of the form name. They always evaluate to the value of the def expression that defines name. If name is not defined, they evaluate to $\bot$.

2.3 Function

Function expressions are expressions of the form (**fn** name value). They evaluate to a function λname.value. When applied, they perform the substitution value[name/argument].

2.4 Impure

Impure expressions are expressions of the form (**impure** value) They evaluate to a process πvalue. When applied, they apply their argument to value. If the result of application is not a process, they evaluate to $\bot$ instead.

2.5 Apply

Apply expressions are expressions of the form (**ap** value argument). If value is a function, it performs application as described in section 2.3. If value is a process, it performs application as described in section 2.4.

2.6 Macro

Macro expressions are expressions of the form (name expression). Before evaluation, they evaluate name and apply it to expression, replacing themselves with the result. The expression is encoded as a coproduct of a symbol and a list as described in section 3.

3 Encodings

```
;;; Functions

(def id (fn x x))

(def compose
  (fn f g
    (fn x
      (ap f g x))))

(def const
  (fn x
    (fn "" x)))

;;; Booleans

(def true
  (fn x
    (fn "" x)))

(def false
  (fn ""
    (fn x x)))

;;; Products

(def pair
  (fn first second
    (fn value
      (ap value first second))))

(def first
  (fn pair
    (ap pair true)))

(def second
  (fn pair
    (ap pair false)))

;;; Coproducts

(def left
  (fn value
    (fn first ""
      (ap first value))))

(def right
  (fn value
    (fn "" second
      (ap second value))))

(def left?
  (fn x
    (ap x (ap const true) (ap const false))))
```

```
(def right?
  (fn x
    (ap x (ap const false) (ap const true))))

;;; Lists

(def nil (ap left false))
(def cons (ap compose right pair))

(def car
  (fn value
    (ap list id left)))

(def cdr
  (fn list
    (ap list id right)))

(def nil? left?)

;;; Natural Numbers

(def 0 ())
(def 0? nil?)
(def inc (ap cons ()))
(def dec cdr)

;;; Chars

(def nat->char id)
(def char->nat id)

;;; Symbols

(def list->symbol id)
(def symbol->list id)

;;; Expressions

(def symbol? left?)
(def list? right?)
```

4 Reference Implementation

4.1 Parser

( )

4.2 Interpreter

( )

4.3 Compiler

( )