Virtual Machine Language T

Kwangkeun Yi

2011, 1996

1 Syntax

This virtual machine's address space is not pre-divided into registers and the main memory. A particular memory architecture is programmed by declaring memory areas (using `AREA`) each of which is addressed by a pair of an area name and an offset in the area. Conventional machines that have registers `r0, r1` and etc. can be mimicked by declaring an area, say, `r` and using locations `r(0), r(1)` and so on.

A single memory location can have either an integer, a string (of any length), a label, or a location.

An abstract syntax of T is:

pgm	::=	*decl*$^+$ *instr*$^+$	program
decl	::=	`AREA` *id*	memory area declaration
instr	::=	`MOVE` *m m*	
	|	`ADD` *m m m*	
	|	`SUB` *m m m*	
	|	`MUL` *m m m*	
	|	`DIV` *m m m*	
	|	`TOZ` *m m*	
	|	`JMP` *m*	
	|	`JMPZ` *m m*	
	|	`JMPN` *m m*	
	|	`LAB` *label*	
	|	`READ` *m*	
	|	`WRITE` *m*	
m	::=	*integer* | `"`*string*`"` | *id* | *label*	
	|	*m*`(`*m*`)`	
	|	*m*`@`$^+$	
id, *label*			alpha-numeric identifier

For example, a summation program in T:

```
AREA A          // sum 0 upto A
AREA S          // S has the result
LAB     START
        READ  A
```

```
        MOVE  0  S
LAB     LOOP
        ADD    S@  A@  S
        SUB    A@  1   A
        JMPZ   A@  EXIT
        JMP    LOOP
LAB     EXIT
        WRITE  S@
LAB     END
```

2 Semantics

Semantic objects

M	$\in$	Mem	$=$	$\mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}$	memory
		Loc	$=$	$\mathit{Area} \times \mathit{Int}$	location
		Val	$=$	$\mathit{Loc} \cup \mathit{Int} \cup \mathit{Lab} \cup \mathit{String}$	value
pc	$\in$	Int			program counter
m	$\in$	Addr			address term
ℓ	$\in$	Lab			label

An interpretation rule:

$$(M, pc) \Longrightarrow (M', pc')$$

indicates that a memory M changes to M' after an instruction at pc is executed, and the next program counter becomes pc'. Notation $M[x \mapsto y]$ indicates a new memory that is equivalent to M except at x, where its value is y.

$$
\begin{array}{lll}
(M, pc : \texttt{MOVE}\ m_1\ m_2) & \Longrightarrow & (M[\mathit{val}(M, m_2) \mapsto \mathit{val}(M, m_1)], pc + 1) \\
(M, pc : \texttt{ADD}\ m_1\ m_2\ m_3) & \Longrightarrow & (M[\mathit{val}(M, m_3) \mapsto \mathit{val}(M, m_1) \oplus \mathit{val}(M, m_2)], pc + 1) \\
(M, pc : \texttt{SUB}\ m_1\ m_2\ m_3) & \Longrightarrow & (M[\mathit{val}(M, m_3) \mapsto \mathit{val}(M, m_1) \ominus \mathit{val}(M, m_2)], pc + 1) \\
(M, pc : \texttt{MUL}\ m_1\ m_2\ m_3) & \Longrightarrow & (M[\mathit{val}(M, m_3) \mapsto \mathit{val}(M, m_1) \otimes \mathit{val}(M, m_2)], pc + 1) \\
(M, pc : \texttt{DIV}\ m_1\ m_2\ m_3) & \Longrightarrow & (M[\mathit{val}(M, m_3) \mapsto \mathit{val}(M, m_1) \oslash \mathit{val}(M, m_2)], pc + 1) \\
(M, pc : \texttt{TOZ}\ m_1\ m_2) & \Longrightarrow & (M[\mathit{val}(M, m_2) \mapsto \mathit{cast}(\mathit{val}(M, m_1))], pc + 1) \\
(M, pc : \texttt{JMP}\ m) & \Longrightarrow & (M, \mathit{pcfy}(\mathit{val}(M, m))) \\
(M, pc : \texttt{JMPZ}\ m_1\ m_2) & \Longrightarrow & (M, \text{if } \mathit{val}(M, m_1) = 0 \text{ then } \mathit{pcfy}(\mathit{val}(M, m_2)) \text{ else } pc + 1) \\
(M, pc : \texttt{JMPN}\ m_1\ m_2) & \Longrightarrow & (M, \text{if } \mathit{val}(M, m_1) < 0 \text{ then } \mathit{pcfy}(\mathit{val}(M, m_2)) \text{ else } pc + 1) \\
(M, pc : \texttt{READ}\ m) & \Longrightarrow & (M[\mathit{val}(M, m) \mapsto \mathit{input}()], pc + 1) \\
(M, pc : \texttt{WRITE}\ m) & \Longrightarrow & (M, pc + 1) \\
(M, pc : \texttt{LAB}\ \ell) & \Longrightarrow & (M, pc + 1) \\
(M, pc : \texttt{LAB END}) & \Longrightarrow & (M, pc)
\end{array}
$$

Auxiliary function $\mathit{val}(M, m)$ indicates a value represented by an address term m at a given memory M:

$$\mathit{val} \colon \mathit{Mem} \times \mathit{Addr} \to \mathit{Val}$$

$val(M, a)$	$= \langle a, 0 \rangle$	aread id
$val(M, x)$	$= x$	integer/string/label
$val(M, m_1\texttt{(}m_2\texttt{)})$	$= val(M, m_1) \oplus val(M, m_2)$	offset address
$val(M, m\texttt{@})$	$= M(val(M, m))$	dereference

Casting function $cast(v)$ casts a location to an integer by removing the area name from the location:

$$cast \colon Loc \to Int$$

$$cast\langle a, z \rangle = z.$$

$v_1 \oplus v_2$ is defined only when

- both v_1 and v_2 are integer. Then the $\oplus$ is the integer addition.
- both v_1 and v_2 are locations whose areas are identical. Then $v_1 \oplus v_2$ is the location in the area whose offset is the sum of v_i's offsets.
- one of them is an integer and the other is a location. Then the result is the location in the same area whose offset is incremented by the integer.

Similarly for $\ominus$,$\otimes$, and $\oslash$.

- "`AREA sp`" declares new memory area `sp`.
- "`MOVE 2 sp`" moves constant 2 to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE "cs" sp`" moves string "cs" to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE sp@ sp`" moves value at location $\langle \texttt{sp}, 0 \rangle$ to location $\langle \texttt{sp}, 0 \rangle$.
- "`MOVE sp(-1)@ sp`" moves value at $\langle \texttt{sp}, -1 \rangle$ to location $\langle \texttt{sp}, 0 \rangle$.

Function $pcfy(\ell)$ returns a label ℓ's program counter value, and $input()$ gets an integer from the outside world.

Execution (semantics) of a program P is the sequence of states starting from an empty memory and the initial program counter at label `START`:

$$([], pcfy(\texttt{START})) \Longrightarrow \cdots$$

Note that the end of the program is when the program counter reaches the `END` label. Therefore any program must have the two labels (`START` and `END`).

□