TML Specification

Pete Gautam

March 17, 2023

In this document, we will define the syntax of the TML, starting with an example. We next analyse the syntax and define execution of a valid TML program on a tape in a similar manner to the execution of a TM.

Consider the following TML program.

```
1  // checks whether a binary number is divisible by 2
2  alphabet = {0, 1}
3  module isDiv2 {
4      // move to the end
5      while 0, 1 {
6          move right
7      } if blank {
8          move left
9          // check last letter is 0
10         if 0 {
11             accept
12         } if 1, blank {
13             reject
14         }
15     }
16 }
```

A program in TML is used to execute on a tape, so the syntax used guides us in executing the program on a tape.

- A valid TML *program* is composed of an *alphabet*, followed by one or more *modules*. In the example above, the alphabet of the program is $\{0, 1\}$, and the program has a single module called `isEven`.
- A module contains one or more *blocks* (a specific sequence of commands). There are two types of blocks- *basic blocks* and *switch blocks*.
- A basic block consists of *basic commands* (*changeto*, *move* or *flow* command). A basic block consists of at least one basic command, but it is not necessary for a basic block to be composed of all the basic commands. If multiple commands are present in a basic block, they must be in the following order- *changeto*, *move* and *flow* command. In the program above, there is are many basic blocks, e.g. at lines 4, 6, 8-9 and 11-12. We do not say that line 8 is a basic block by itself; we want the basic block to be as long as possible.

- A *switch block* consists of cases (*if* or *while* commands), each of which corresponds to one or more letters. A switch block must contain precisely one case for each of the letter in the alphabet, including the `blank` letter. The first block within a case block cannot be another switch block. In the program above, there is a switch block at lines 3-14 and a nested switch block at lines 7-13.
- The body of an *if* command can be composed of multiple blocks. These blocks can be both basic blocks and switch blocks. We can see this at lines 5-13; the *if* block has a basic block at line 6 and then a switch block.
- The body of a *while* command must be composed of a single basic block. The basic block cannot have a *flow* command. This is because when we execute a *while* block, the next block to run is the switch block it is in; we cannot accept, reject or go to another module.
- A switch block must be the final block present; it cannot be followed by a basic block.

$$
\begin{aligned}
\textit{program} &= \textit{alphabet}\ \textit{module}^{+} \\
\textit{alphabet} &= \texttt{alphabet =}\ \{\ \textit{seq-val}\ \} \\
\textit{module} &= \texttt{module}\ \textit{id}\ \{\ \textit{block}^{+}\ \} \\
\textit{block} &= \textit{basic-block} \mid \textit{switch-block} \\
\textit{switch-block} &= \textit{case-block}^{+} \\
\textit{case-block} &= \textit{if-block} \mid \textit{while-block} \\
\textit{if-block} &= \texttt{if}\ \textit{seq-val}\ \{\textit{block}^{+}\} \\
\textit{while-block} &= \texttt{while}\ \textit{seq-val}\ \{\ \textit{core-com}^{+}\ \} \\
\textit{basic-block} &= (\textit{core-com} \mid \textit{flow-com})^{+} \\
\textit{core-com} &= \texttt{move}\ \textit{direction} \mid \texttt{changeto}\ \textit{value} \\
\textit{flow-com} &= \texttt{goto}\ \textit{id} \mid \textit{terminate} \\
\textit{terminate} &= \texttt{reject} \mid \texttt{accept} \\
\textit{direction} &= \texttt{left} \mid \texttt{right} \\
\textit{seq-val} &= (\textit{value}\texttt{,})^{*}\ \textit{value} \\
\textit{value} &= \texttt{blank} \mid \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \ \ldots\ \mid \texttt{z} \mid \texttt{0} \mid \texttt{1} \mid \ \ldots\ \mid \texttt{9} \\
\textit{id} &= (\texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \ \ldots\ \mid \texttt{z} \mid \texttt{A} \mid \texttt{B} \mid \texttt{C} \mid \ \ldots\ \mid \texttt{Z})^{+}
\end{aligned}
$$

Figure 1: The EBNF of the TML.

The EBNF of the TML is Figure 1.

We will now consider how to execute a tape on a valid TML program. Let P be a TML program with alphabet Σ and let T be a tape on Σ. We execute P on T inductively, as follows:

- At any point during execution, we maintain 3 objects- a tape on Σ, a block of P and the tapehead index.
- At the start, the tape is T; the tapehead index is 0; and the block is the first block in the first module in P.
- At some point during the execution, assume that we have the tape S, tapehead index j, with tapehead value $T(j) = t$, and a block b. We define the next triple as follows:
  - if b is a *switch* block, we take the first block from the case corresponding to the tapehead value- because the program is valid, this is a basic block; we will now refer to this block as b.
  - if b has a *changeto* `val` command, the next tape T' is given by

$$T'(x) = \begin{cases} \texttt{val} & x = i \\ T(x) & \text{otherwise.} \end{cases}$$

    If the *changeto* command is missing, then the tapehead $T' = T$.
  - if b has a *move* `dir` command, the next tapehead index is given by:

$$i' = \begin{cases} i+1 & \texttt{dir} = \texttt{right} \\ i-1 & \texttt{dir} = \texttt{left}. \end{cases}$$

    If the *move* command is missing, then $i' = i - 1$.
  - we either terminate or determine the next block b' to execute (in decreasing precedence):
    - if the block is the body of a while case block, then the next block $b' = b$, i.e. we execute this switch block again (not necessarily the same case block);
    - if the block contains a terminating *flow* command, execution is terminated and we return the terminated state (`accept` or `reject`);
    - if the block contains a *goto* `mod` command, then b' is the first block of the module `mod`;
    - if the block is not the final block in the current module, then b' is next block in this module;
    - otherwise, execution is terminated and we return the state `reject`.

  If execution is not terminated, execution continues with the next triplet.

We will now illustrate this process. We execute the program `isEven` on the tape at Figure 2.

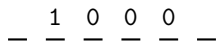
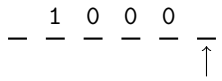


Figure 2: A TM tape on $\{0, 1\}$.

- Initially, the tape is the given tape; the current block is at lines 5-15; and the tapehead index is 0, with value 1.
- Since the tapehead value is `1`, the basic block to be executed is lines 5-7. Hence,
  - the tape remains unchanged;
  - the current block is still at lines 5-15; and
  - and the tapehead index becomes 1, with value `0`.
- The transition for `0` and `1` are the same with respect to the current block. This means that we keep moving to the right until we end up at a blank symbol. At that point, the following is the state of the tape:

  The arrow points at the tapehead value. We are still executing the same block, and the tape has not been altered.
- Now, since the tapehead value is `blank`, we move to the left. Moreover, the current block is at lines 10-14. The tape has still not been changed. The current value is now 0.
- The tapehead value is currently `0`. So,
  - the tape value changes to `blank`;
  - we have reached the `accept` command; and
  - the tapehead pointer move to the left (by default), to index 2.

  Hence, execution terminates, with result accept and the following tape state: