TML Equivalence Proof

Pete Gautam

October 5, 2022

Abstract

This document proves the equivalence of the TML with TMs. We do this by constructing an equivalent TML program for a TM, and vice versa. In the first section, we define complete TML programs, which are very close to TMs, and describe how they can be mapped into TMs.

1 Complete TML programs

Complete programs in the TML are a specific type of TML programs that are very detailed and obey different properties. As such, it is very easy to construct the TM corresponding to a complete program. In this section, we will build to the definition complete programs from complete blocks and modules.

Definition 1.1. Let P be a valid TML program, and let B be a basic block in P. We say that B is a *complete block* if it is composed of all the 3 commands: a *changeto* command, a *move* command, and a *flow* command. If the *flow* command is missing, we say that B is a *partially complete block*.

Complete blocks contain all the information required to transition from one state to another. This is because of the following:

- A complete block lists the next value of the tapehead; we assume the original value of the tapehead to be known outwith the block.
- A complete block states which direction the tapehead is moving- left or right.
- A complete block determines precisely what the next state is for the block- it is either a terminating state or we are going to the initial block of another module.

We can convert any basic block with a *flow* command to a complete block by adding the default commands (i.e. moving left and changing the tapehead value to the current value). Equally, it is quite straightforward to convert a complete block into a subpart of a Turing Machine.

Example 1.2. Consider the following complete block.

```
changeto a
move right
goto s2
```

If we are at the state s_1, and the block applies when the tapehead value is b, then the following is the corresponding subpart of the Turing Machine:

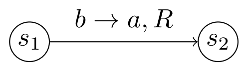

Figure 1: The TM subpart of the given Turing Machine complete block.

Using complete and partially complete blocks, we can define complete modules.

Definition 1.3. Let P be a valid TML program, and let M be a module in P. We say that M is a *complete module* if all of the following hold:

- it is composed of a single switch block;
- the body of each *if* command is a complete block; and
- the body of each *while* command is a partially complete block.

Remark 1.4. For a *while* command, the body must be partially complete because the corresponding edge in the TM is always a loop.

It is quite easy to map a single module to a sub-Turing machine. It precisely corresponds to having an initial state and edges going to other states as dictated by each case. Because the program is valid, we know that there is a case for each letter in the alphabet, including the `blank` letter.

Example 1.5. Consider the following complete module.

```
module basic {
    switch tapehead {
        while b {
            changeto b
            move right
        } if a, blank {
            changeto blank
            move left
            reject
        }
    }
}
```

If the alphabet is composed only of a and b, then the corresponding sub-TM is the following:

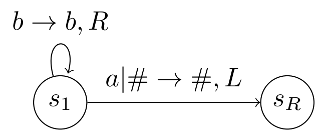

Figure 2: The sub-TM of the given Turing Machine complete module.

Remark 1.6. In the example above, we consider the state s_1 to be the *corresponding state* of the module. For every complete module, there is precisely one corresponding state.

Now, we can define complete programs.

Definition 1.7. Let P be a TML program. We say that P is *complete* if it is composed of one or more complete modules. We also require every *goto* command in a complete block to refer to an existing module.

Remark 1.8. The second condition (called *valid reference*) is required for any TML program.

Remark 1.9. In general, a complete program is not composed of a single complete module. We will see later that a relatively simple module can be broken down into a couple of complete modules, each of which refer to each other.

Every module in the program can be converted to a state, along with directed edges to other states. If the program is complete, then we ensure that the states connect to form a valid TM.

Example 1.10. Consider the following complete program.

```
alphabet = {"a", "b"}
module isDivTwo {
    switch tapehead {
        while 0 {
            changeto 0
            move right
        } while 1 {
            changeto 1
            move right
        } if blank {
            changeto blank
            move left
            goto isDivTwoCheck
        }
    }
}
module isDivTwoCheck {
    switch tapehead {
        if 0 {
            changeto 0
            move left
            accept
        } if 1, blank {
```

```
            changeto blank
            move left
            reject
        }
    }
}
```

Then, the corresponding TM is the following:

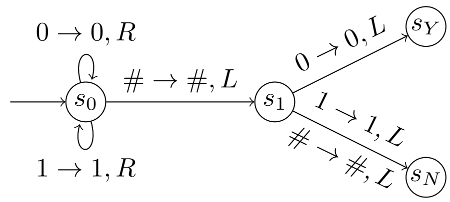

Figure 3: The TM of the given TML program. The state s_0 corresponds to the module `isDivTwo` and the state s_1 corresponds to the module `isDivTwoCheck`.

Remark 1.11. In the example above, we converted a complete TML program into a TM. It is equally possible to convert a TM into a complete TML program.

Example 1.12. Consider the following TM:

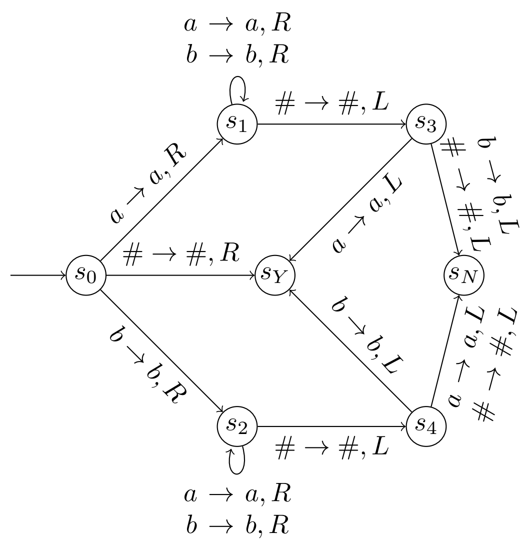

Figure 4: A Turing machine

Then, its corresponding TML program is:

```
alphabet = {"a", "b"}
module startsAndEndsSame {
```

```
    switch tapehead {
        if blank {
            accept
        } if a {
            changeto a
            move right
            goto startsAndEndsSameMoveA
        } if b {
            changeto b
            move right
            goto startsAndEndsSameMoveB
        }
    }
}
module startsAndEndsSameMoveA {
    switch tapehead {
        while a {
            changeto a
            move right
        } while b {
            changeto b
            move right
        } if blank {
            changeto blank
            move left
            goto startsAndEndsSameCheckA
        }
    }
}
module startsAndEndsSameCheckA {
    switch tapehead {
        if a {
            changeto a
            move left
            accept
        } if b {
            changeto b
            move left
            reject
        } if blank {
            changeto blank
            move left
            reject
        }
    }
}
module startsAndEndsSameMoveB {
    switch tapehead {
        while a {
            changeto a
            move right
        } while b {
            changeto b
            move right
        } if blank {
            changeto blank
            move left
```

```
            goto startsAndEndsSameCheckB
        }
    }
}
module startsAndEndsSameCheckB {
    switch tapehead {
        if a {
            changeto a
            move left
            reject
        } if b {
            changeto b
            move left
            accept
        } if blank {
            changeto blank
            move left
            reject
        }
    }
}
```

This is a complete program since TMs always include the required commands corresponding to *if* and *while* commands.

Remark 1.13. Although a TML program need not be complete, any valid TML program is equivalent to a complete one. So, a TML program that is not be complete is just a compact representation of its complete version.