MCS-265: The Theory of Computation Handout #T1
San Skulrattanakulchai
Gustavus Adolphus College April 8, 2014

Designing 1-tape Deterministic Turing Machines

A TM corresponds to a computer program. We will develop a pseudocode language for designing TM's. It has the following features.

- There are four basic statements:
  - *move R*
  - *move L*
  - *write X & move R*
  - *write X & move L*

  where X is a tape symbol. A basic statement corresponds to one TM transition minus the state change. That is, a statement can move one cell either left or right, after possibly writing a new symbol X on the current tape cell.
- The special basic statement *accept* means the TM enters the final (accepting) state.
- The special basic statement *die* corresponds to the fact the TM enters the rejecting state.
- A sequence of statements can be turned into one statement by putting braces around them.
- There are 3 control flow constructs: **while**, **if**, and **goto**. The syntax is

  while ⟨`condition`⟩ **do** ⟨`statement`⟩,

  if ⟨`condition`⟩ **then** ⟨`statement`⟩ [**else** ⟨`statement`⟩],

  and

  goto ⟨`label`⟩

respectively. The ⟨`condition`⟩ in the while and if statements is a disjuntion involving (one or more) input symbols. These three constructs mean just like their counterparts in C or java.

- A comment starts with // and takes effect till the end of line.
- There is no statement separator. Indentation is used to distinguish statement levels.

To translate a program written in pseudocode into a TM, we first introduce TM states as labels for pseudocode statements at appropriate places. We then manually translate each pseudocode statement into TM transitions.

An example will make this clear. Say we want to design a TM to recognize the language of strings over $\Sigma = \{0, 1\}$ containing an equal number of 0's and 1's. We will use $\Gamma = \{0, 1, X, Y, B\}$ as our tape alphabet. (B stands for the blank symbol.) Our algorithm works by first finding the leftmost 0 or 1, and turning it into an X. It then scans right to find the first opposite symbol (1 for 0, and 0 for 1) and then turning it into a Y. It then scans left to find the rightmost X and repeats this process. The algorithm accepts if every symbol turned into X has a matching opposite symbol turned into Y. It rejects otherwise. Here is the pseudocode.

```
q_0:    while X or Y do
            move R
        if B then
            accept
        else if 0 then {
            write X & move R
            while 0 or Y do
                move R
            if 1 then
                write Y & move L
            else
                die // is seeing a B, which means input has more 0's than 1's
            while 0 or Y do
                move L
            // must now be seeing an X
            goto q_0
        } else if 1 then {
            // symmetric, similar to the 0 case, i.e.,
            // same as above, with roles of 0 and 1 interchanged
        }
```

We need to prove that this algorithm works correctly. Let w denote the non-blank portion of the input tape. We make the following claims.

1. At any time, $w \in \{X, Y, 0, 1\}^*$.

2. Whenever the TM is at q_0, the number of occurrences of X's within w equals the number of occurrences of Y's within w.

3. At any time, in any prefix of w, the number of occurrences of X's is at least the number of occurrences of Y's.

4. At any time, no 0 or 1 occurs to the left of the rightmost X.

5. There exists a constant k such that whenever the TM is at q_0, the difference

between the number of occurrences of 0's within w and the number of occurrences of 1's within w equals k.

6. Each time the TM reenters q_0, the number of occurrences of 0's and 1's within w each decreases by 1.

7. This TM always halts, either because it dies or enters an accepting state.

8. If the TM accepts, 0 occurs as many times as 1 in the input. If the TM dies, 0 does not occur as many times as 1 in the input.

These claims should be proved in the order listed. Many of these claims can be proved by induction on the number of times the TM enters q_0. The last two claims prove correctness of the TM.

We will now translate the pseudocode into TM transitions. First, by inspecting the code, we insert labels as follows.

```
q_0:    while X or Y do
            move R
        if B then
            accept
        else if 0 then {
        q_1: write X & move R
            while 0 or Y do
                move R
            if 1 then
                write Y & move L
            else
                die // is seeing a B, which means input has more 0's than 1's
        q_2: while 0 or Y do
                move L
            // must now be seeing an X
            goto q_0
        } else if 1 then {
        q_3: write X & move R
            while 1 or Y do
                move R
            if 0 then
                write Y & move L
            else
                die // is seeing a B, which means input has more 1's than 0's
        q_4: while 1 or Y do
                move L
            // must now be seeing an X
            goto q_0
        }
```

Next, we translate each pseudocode statement into transitions to get this table.

δ	0	1	X	Y	B
q_0	(q_1, X, R)	(q_3, X, R)	(q_0, X, R)	(q_0, Y, R)	(q_f, B, R)
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	(q_4, Y, L)	$(q_3, 1, R)$	–	(q_3, Y, R)	–
q_4	–	$(q_4, 1, L)$	(q_0, X, R)	(q_4, Y, L)	–
q_f	–	–	–	–	–

This TM has 6 states. Upon closer examination (of both the transition table and the pseudocode), we find that states q_2 and q_4 can safely be merged (Why?). Merging them into q_2 gives the following equivalent TM with 5 states.

δ	0	1	X	Y	B
q_0	(q_1, X, R)	(q_3, X, R)	(q_0, X, R)	(q_0, Y, R)	(q_f, B, R)
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$	(q_0, X, R)	(q_2, Y, L)	–
q_3	(q_2, Y, L)	$(q_3, 1, R)$	–	(q_3, Y, R)	–
q_f	–	–	–	–	–