Turing Machines for Complicated Tasks

Our primary interest is not in programming TMs, but in determining whether or not a TM can be programmed for a given task.

We use block diagrams and pseudocode to describe TMs at a high-level. Sometimes, we implement parts of the description, but usually we settle for a convincing argument that the implementation is just a matter of details.

How to Construct an Adder

In my adding TM, the input consists of two binary numbers separated by a ‘:’. Here are some intermediate stages of the tape.

```plaintext
__1011:1001__
__1011:1001_0
__101::100_10
__10:::10_100
__1::::1_0100
__:::::_10100
________10100
```
Here are brief explanations of the states:

q0,q1 Put a 0 on the right.
q2,q3,f0 Detect when it’s finished.
q2-q5 Extract last bit of first input.
a0-a1 This bit is 0 or 1.
b0-b1 Extract last bit of second input.
c0-c2 Total of two bits is 0, 1, or 2.
Add to previous carry bit.
d0-d1 Next carry bit is 0 or 1.
How to Find a Substring

In my substring TM, the input consists of two bit strings separated by a ':'. Here are some intermediate stages of the tape.

__110:01110__
__b10:a1110__
__110:a1110__
__bba:abbb0__
__110:ab110__
__bba:abbb0a__

Here are brief explanations of the states:

- **q0**: Identify next symbol to match.
- **a0,a1**: Match 0 or 1, moving in left part.
- **b0,b1**: Match 0 or 1, moving in right part.
- **r0,r1**: Found a match, get next symbol.
- **s0-s5**: Match failed,
- **s0-s2**: Reverse symbol in left part.
- **s3-s5**: Reverse symbol in right part.
Block Diagrams and Pseudocode

\[ x, y \rightarrow z := 0 \rightarrow x = 0? \]
\[ \text{T} \rightarrow \text{T} \rightarrow z \]
\[ \text{F} \rightarrow b = 1? \]
\[ \text{T} \]
\[ b := x \mod 2 \]
\[ \text{F} \rightarrow x := (x-b)/2 \rightarrow y := y*2 \]
\[ \text{F} \rightarrow \text{F} \rightarrow \]
\[ b = 1? \]
\[ \text{T} \rightarrow z := z + y \]
Sequencing of Turing Machines

If we want TM \( M_1 \) followed by TM \( M_2 \), then:

1. Rename the states so that \( M_1 \) and \( M_2 \) have no name in common.

2. Add a transition from \( M_1 \)'s final state to \( M_2 \)'s start state.

Using other Turing Machines as Subroutines

To have TM \( M_1 \) use TM \( M_2 \) as a subroutine,

1. Rename the states so that \( M_1 \) and \( M_2 \) have no name in common.

2. \( M_1 \) writes the input \( x \) for \( M_2 \) as \( \#x \) with infinite blanks to the right.

3. \( M_2 \) is rewritten to not use any space left of the \( \# \).

4. \( M_2 \) is rewritten so the output \( y \) is after the \( \# \) with infinite blanks to the right.