Solution Set 7

DATE: 24 NOVEMBER, 1992

1. Multiply on a Turing Machine:

$$\boxed{b}\,1^{x+1}\,b\,1^{y+1}\,b^* \Rightarrow \boxed{b}\,1^{xy+1}\,b^*$$

We build this up from subroutines. In class we showed how to have a subroutine which scans left or right for double-blank. We first describe how to add a number to its neighbor provided that a one exists to the left:

$$1\,\boxed{b}\,1^{x+1}\,b\,1^{y+1}\,b^* \Rightarrow 1\,\boxed{b}\,1^{x+1}\,b\,1^{x+y+1}\,b^*$$

Additionally, we will never move the head leftwards of the leftmost character in the picture.

(a) Repeat until the head discovers $\boxed{b}\,1\,b$, that is, until x becomes 0,
- Move right.
- Put a blank.
- Scan right for a double-blank.
- Put a one
- Scan left for a double-blank.

(b) Once the termination-condition is satisfied, write 1's while moving left until a 1 is found.

(c) Now move once right and write a blank.

The overall Turing Machine works as follows:

(a) Check if either x or y is zero. If so, clear the tape, write zero and stop.

(b) Else, effectuate the following transformation:

$$\boxed{b}\,1^{x+1}\,b\,1^{y+1}\,b^* \Rightarrow \boxed{b}\,1^{x}\,b\,1^{y+1}\,b\,1^{y+1}\,b^*$$

This can be accomplished as a small variation on the addition subroutine outline above.

(c) Reposition the head:

$$\boxed{b}\,1^x\,b\,1^{y+1}\,b\,1^{y+1}\,b^* \Rightarrow b\,1^x\,\boxed{b}\,1^{y+1}\,b\,1^{y+1}\,b^*$$

(d) Apply this procedure until the head discovers $b\,1\,\boxed{b}$, that is, until x becomes 0,

- Decrement the number to the right of the head:
  - Search left to double-blank.
  - Move right.
  - Write a blank.
  - Search right to single-blank.
- Add the number directly to the left of the head to its left neighbor. Use the above described subroutine.

(e) Clean-up: move left, write a blank, move right twice. Until the head is above a blank, write a blank and move right.

2. Simulate a TM with a while-program. We need a set of variable definitions and macros which simulate the basic Turing Machine functions: move right, move left, write a one, write a blank and sense the symbol under the head. We represent the tape's contents as three numbers: *left-tape, right-tape* and *under-head*. Suppose the tape is as displayed:

$$\ldots\, l_3\, l_2\, l_1\, l_0\, \boxed{h}\, r_0\, r_1\, r_2\, r_3\, \ldots$$

where h is either 1 or b and both the l_i and the r_i form semi-infinite sequences $i = 0, 1, 2, \ldots$ which are all blanks for large enough i. Then,

$$\begin{aligned} \textit{left-tape} &= \sum_{i=0}^{\infty} l_i 2^i, \\ \textit{right-tape} &= \sum_{i=0}^{\infty} r_i 2^i, \\ \textit{under-head} &= h, \end{aligned}$$

where a blank is interpreted arithmetically as a 0. To sense the tape head, just look at the value of *under-head*. To set the tape under the head to 1 or blank, change the value of *under-head*. To move right, update the variables,

```
left-tape := 2 * left-tape + h ;
h := right-tape mod 2 ;
right-tape := right-tape div 2 ;
```

To move left, interchange the words "right" and "left" above.

Given these macros, we transform a TM to a while-program by numbering its states $0, \ldots, n$, where 0 is the halt state and 1 is the start state. The variables $X1, \ldots, Xk$ are written to the simulated tape, the variable *state* is set to 1 and the while program executes:

```
while state<>0 do
    case state of
     1: if under-head=1 then begin
          {update state and tape}
        else
          {update state and tape}
     2: ...
     .
     .
     .
    end case ;
```

The TM state-transition table is encoded in the case-statement. When the loop is exited, the simulated tape is written to $X1$.

What is the time for this simulation? It would be convenient for the theory of complexity if the simulation was polynomial time. However, this isn't so. The unary encoding of x becomes the value $2^{x+1}-1$ inside the while-program. Just to get some variable inside the while-program to go from a representation of $x \geq 1$ to a representation of $2x$ at least:

$$2^{2x+1} - 1 - (2^{x+1} - 1) \geq 4^x$$

successor operations will be required! On the other hand, a Turing Machine can perform multiplication by two in time polynomial in the input.

As an exercise, try to construct a polynomial-time simulation of a Turing Machine by a while-program or prove that this is impossible.

3. Write a program that computes itself. Here is one in Pascal:

```
program pm(input,output);const a='program pm(input,
output);const a=;begin write(substr(a,1,33)+chr(39)
+a+chr(39)+substr(a,34,66))end.';begin write(substr
(a,1,33)+chr(39)+a+chr(39)+substr(a,34,66))end.
```

A terse C solution was offered by Bentley Hargrave:

```
char*s="char*s=%c%s%c;main(){printf(s,34,s,34,10);
}%c";main(){printf(s,34,s,34,10);}
```

The line breaks shown in these programs are not actually part of their text.