5 Encoding Turing machines

The C program for applying Turing machines takes two files, one being a Turing machine presented as a list of quintuples, and the other being sample data (the data is a set of inputs, one per line).

Example Turing machine. The following is a Turing machine to increment binary numbers (possibly the same machine was shown in the previous section).

```
q0 0 0 R q0
q0 1 1 R q0
q0 B B L q1
q1 1 0 L q1
q1 0 1 L q2
q1 B 1 L q2
q2 0 0 L q2
q2 1 1 L q2
q2 B B R q3
```

What we have here is a set of ASCII strings of a certain format; it is a kind of programming language.

And here is a sample run of this machine (three separate tests)

```
input: 0
output: 1

input:
output: 1

input: 1110000110111
output: 1110000111000
```

5.1 Encoding Turing machines as bitstrings

If we could encode Turing machines as bitstrings, then we could possibly build an interpreter for Turing machines, *where the 'interpreter' is itself a Turing machine.*

To encode a Turing machine as a bitstring is easy. There are millions of ways to choose an encoding: here is one.

It is assumed that the input alphabet is binary, the tape alphabet is $X_0, \ldots, X_t$, where $X_0 = 0, X_1 = 1, X_2 = B$, and the other tape symbols are in arbitrary order.

It is assumed that the state symbols are q_i, and q_0 is the initial state.

A quintuple

$$q_i X_j X_k L q_\ell$$

is encoded as

$$1\ 10^{i+1}1\ 10^{j+1}1\ 10^{k+1}1\ 101\ 10^{\ell+1}1\ 1$$

$$q_i X_j X_k R q_\ell$$

is encoded as

$$1\ 10^{i+1}1\ 10^{j+1}1\ 10^{k+1}1\ 1001\ 10^{\ell+1}1\ 1$$

Notice that every quintuple is 'framed' between two 1s. The full encoding of the Turing machine is the bitstring

$$111\ Q_1\ \ldots\ Q_k\ 111$$

Or rather, an encoding: the quintuples can be listed in any order. For example, the above incrementing machine is encoded as

```
111
110110110110011011
110110011001100110 11
110110001100011011 0011
11001100110011011 0011
11001101100110110 0011
11000110011001101100011
11000110011001101100011
11000110100110110 0011
11000110011001101100011
11000110001100011001100011
111
```

It can be checked by decoding

```
q0 0 0 R q0
q0 1 1 R q0
q0 B B L q1
q1 1 0 L q1
q1 0 1 L q2
q1 B 1 L q2
q2 0 0 L q2
q2 1 1 L q2
q2 B B R q3
```

The quintuples are framed $111\ldots111$ because only at the beginning and the end are there more than four 1s in succession. The important consequence is:

(5.1) Lemma *If x is any bitstring, and x can be 'factorised' as the product of two strings y and z concatenated, i.e., $x = yz$, and y is a valid encoding of a Turing machine, then the factorisation is unique.*

Put another way: if there exist bitstrings y_1, z_1, y_2, z_2 such that $x = y_1z_1 = y_2z_2$ and y_1 and y_2 are both valid encodings of Turing machines, then $y_1 = y_2$ and $z_1 = z_2$. ∎

Actually, X_i is not the only permitted style of tape symbol. The following code (which is related to the Halting Problem) doubles its input, using extra tape symbols. It can still be encoded: the extra tape symbols are indexed in the order of appearance.

```
q0 0 s R q1  q1 0 0 R q1  q1 1 1 R q1  q1 s s R q1
q1 t t R q1  q1 u u R q1  q1 v v R q1  q1 B u L q2
q2 0 0 L q2  q2 1 1 L q2  q2 u u L q2  q2 v v L q2
q2 s s R q0  q2 t t R q0  q0 1 t R q3  q3 0 0 R q3
q3 1 1 R q3  q3 s s R q3  q3 t t R q3  q3 u u R q3
q3 v v R q3  q3 B v L q4  q4 0 0 L q4  q4 1 1 L q4
q4 u u L q4  q4 v v L q4  q4 s s R q0  q4 t t R q0
q0 u 0 R q5  q0 v 1 R q5  q5 u 0 R q5  q5 v 1 R q5
q5 B B L q6  q6 0 0 L q6  q6 1 1 L q6  q6 s 0 L q6
q6 t 1 L q6  q6 B B R q7
```

Encoded:

```
111
1101101100001100110011
1100110110110011001l
1100110011001100110011
1100110000110000110011001l
1100110000011000001100110011
1100110000001100000011001l0011
1100110000000110000000110011001l
1100110001100000011011000l1
1100011011011011000l1
etcetera
```

Decoded:

```
q0 0 X3 R q1   q1 0 0 R q1    q1 1 1 R q1
q1 X3 X3 R q1  q1 X4 X4 R q1  q1 X5 X5 R q1
q1 X6 X6 R q1  q1 B X5 L q2   q2 0 0 L q2
q2 1 1 L q2    q2 X5 X5 L q2  q2 X6 X6 L q2
q2 X3 X3 R q0  q2 X4 X4 R q0  q0 1 X4 R q3
q3 0 0 R q3    q3 1 1 R q3    q3 X3 X3 R q3
q3 X4 X4 R q3  q3 X5 X5 R q3  q3 X6 X6 R q3
q3 B X6 L q4   q4 0 0 L q4    q4 1 1 L q4
q4 X5 X5 L q4  q4 X6 X6 L q4  q4 X3 X3 R q0
q4 X4 X4 R q0  q0 X5 0 R q5   q0 X6 1 R q5
q5 X5 0 R q5   q5 X6 1 R q5   q5 B B L q6
q6 0 0 L q6    q6 1 1 L q6    q6 X3 0 L q6
q6 X4 1 L q6   q6 B B R q7
```

And applied:

```
% tm double.tu 11011
input: 11011
output: 1101111011
```