**§6. Universal Turing machines.** The quality that distinguished the first digital computers[18] from all previous machines was *flexibility*. By placing the operation of the machine under the control of a *program*, the digital computer could be adapted to a wide range of applications.

Until now, we have thought of a Turing machine as being a fixed piece of hardware performing a specialized function. However, if the Turing machine really is a sound model of computation, it should be possible to demonstrate that it can act as a stored program machine, where the program is regarded as an input, rather than hard-wired. That is the purpose of this note. We shall construct a Turing machine $M_u$ that takes as input a *description* of a Turing machine $M$ and an input word $x$, and simulates the computation of $M$ on input $x$ (cf. the program descriptions in §1.3 of Note 1). A machine such as $M_u$ that can simulate the behaviour of an arbitrary Turing machine is called a *universal Turing machine*.

To simplify our task, we shall make two assumptions about the machine $M$ that the universal machine is required to simulate. The more significant simplification is that the input and tape alphabets of $M$ are fixed in advance; specifically the input alphabet of $M$ is $\Sigma = \{\texttt{0}, \texttt{1}\}$, and the tape alphabet is $\Gamma = \{\texttt{0}, \texttt{1}, \not{b}\}$. We shall refer to such a machine as a *binary* Turing machine. The less significant simplification is that we do not allow $M$ the luxury of a final state. Instead, we assume the existence of some other criterion for acceptance; for example, we might agree that $M$ has accepted its input if $M$ gets stuck in a certain state.

**§6.1. Encoding Turing machines.** Let $M = (Q, q_I, \delta)$ be a Turing machine of the above form; $M$ is simply a triple because $\Sigma$, $\Gamma$, and $\not{b}$ are fixed in advance, and the final state is absent. In order to present $M$ as an input to the universal machine $M_u$, it is first necessary to *encode* $M$ as a word over some *fixed* alphabet. The rules (a)–(e) that follow describe a suitable encoding of $M$ as a word over the alphabet $\{\texttt{0}, \texttt{1}, \texttt{B}, \texttt{*}\}$.

(a) Encode the states of $M$ as elements of $\{\texttt{0}, \texttt{1}\}^k$, where $k$ is some suitably chosen integer, e.g., $k = \lceil \lg |Q| \rceil$. We insist that the initial state $q_I$ receives code $\texttt{0}^k$, but otherwise the assignment of codes to states is arbitrary.

(b) Encode the tape symbols $\texttt{0}$, $\texttt{1}$, and $\not{b}$ as $\texttt{0}$, $\texttt{1}$, and $\texttt{B}$, respectively. (The point here is that the true blank symbol $\not{b}$ must not appear in the encoding.)

(c) Encode the directions *left* and *right* by $\texttt{0}$ and $\texttt{1}$, respectively.

(d) Recall that the transition function $\delta$ of $M$ may be viewed a set of quintuples. Encode a quintuple $(q, s, q', s', d)$ as a string $\langle q\rangle\langle s\rangle\langle q'\rangle\langle s'\rangle\langle d\rangle$ of length $2k+3$;

[18] (more accurately, stored program machines)

here $\langle q\rangle$, $\langle s\rangle$, $\langle q'\rangle$, $\langle s'\rangle$, and $\langle d\rangle$ are the codes for $q$, $s$, $q'$, $s'$, and $d$, as defined in rules (a)–(c) above.

(e) Suppose that the transition function $\delta$ is specified by the $m$ tuples $t_0, t_1, \ldots, t_{m-1}$. Encode the machine $M$ itself as the word

$$\langle M\rangle = \langle t_0\rangle{*}\langle t_1\rangle{*}\langle t_2\rangle{*}\cdots{*}\langle t_{m-1}\rangle,$$

where $\langle t_i\rangle$ denotes the encoding of tuple $t_i$ according to rule (d).

By way of example, consider the machine $M_{\text{pred}}$ with states $Q = \{q_0, q_1, q_2\}$, initial state $q_0$, and transition function specified by the set of quintuples

$$\{(q_0, \mathtt{0}, q_0, \mathtt{0}, R), (q_0, \mathtt{1}, q_0, \mathtt{1}, R), (q_0, \not b, q_1, \not b, L), (q_1, \mathtt{0}, q_1, \mathtt{1}, L), (q_1, \mathtt{1}, q_2, \mathtt{0}, R)\}.$$

(On input $x \in \{\mathtt{0},\mathtt{1}\}^*$, the machine $M_{\text{pred}}$ computes $y = x - 1$, where $x$ and $y$ are interpreted as binary numbers, note that if $x$ represents the number 0 then the machine falls off the left end of its tape.) The encoding $\langle M_{\text{pred}}\rangle$ of $M_{\text{pred}}$ obtained by applying rules (a)–(e) is

`0000001*0010011*00B01B0*0100110*0111001.`

Note that we have set $k = 2$, and made the obvious correspondence between the states of $M_{\text{pred}}$ and the binary numbers `00`, `01`, and `10`.

**§6.2. A universal Turing machine.** The universal Turing machine $M_u$ will now be described. The input alphabet of $M_u$ is $\{\mathtt{0}, \mathtt{1}, \mathtt{B}, {*}, \$, \hat{}\}$. Suppose it is desired to simulate the computation of machine $M$ on input $x \in \{\mathtt{0},\mathtt{1}\}^*$. The pair $(M, x)$ would then be presented to the universal machine $M_u$ in the following format:

$$\$\mathtt{0}^{k+1}{*}\langle M\rangle\$\hat{}\,x.$$

Thus, to simulate $M_{\text{pred}}$ on input `1011`, we would initialize the tape of $M_u$ to read as follows:

`$000*0000001*0010011*00B01B0*0100110*0111001$^1011.` $(*)$

The simulation proceeds in a succession of *cycles*; in a single cycle, the simulated machine progresses by one step. We shall work through a single cycle of $M_u$, using the simulation of $M_{\text{pred}}$ as an example.

Suppose the tape of $M_u$ is initialized as shown in $(*)$. After five cycles, the contents of the tape of $M_u$ will in fact be

`$01B*0000001*0010011*00B01B0*0100110*0111001$101^1.`

The tape contents can be interpreted as follows. The first $k$ symbols following the leftmost `$` encode the current state of the simulated machine, in this case $q_1$. The next symbol can be ignored. Then, sandwiched between an asterisk and a dollar symbol, is the encoding of the simulated machine. Finally come the tape contents of the simulated machine, with a caret mark `^` indicating the position of the tape head. (In this instance, the tape contents are `1011`, and the head is scanning the final `1`.)

A cycle of $M_u$ naturally breaks down into six phases, which are now described.

**Reading the scanned symbol.** The machine $M_u$ locates the caret mark, and remembers the symbol (`0`, `1`, or $\not b$) that appears immediately to its right. $M_u$ then moves left and writes the corresponding code (`0`, `1`, or `B`) immediately to the left of the leftmost asterisk. In our example, the scanned symbol is `1`:

```
$011*0000001*0010011*00B01B0*0100110*0111001$101^1.
```

This operation is accomplished by the states `read0`, `read1`, ..., `read6` with their associated transitions in the machine `univ.tm` that you can download from the course web page. (Note: that machine is slightly different from the one described here, with an additional tape symbol to make it simpler to write down.)

**Locating the quintuple.** The string of symbols between the dollar and the first asterisk is now $\langle q\rangle\langle s\rangle$, where $q$ is the state of the simulated machine, and $s$ is the scanned symbol. The tuple that governs the next transition (if any) is the one that has $\langle q\rangle\langle s\rangle$ as a prefix (in this case, the final tuple in the encoding). The machine $M_u$ searches right along the tape until it locates the prefix in question, making the substitutions `0` $\to$ `X`, `1` $\to$ `Y`, and `B` $\to$ `Z` as it goes. If the prefix is not found, $M_u$ halts. In our example, the tape now reads:

```
$011*XXXXXXY*XXYXXYY*XXZXYZX*XYXXYYX*XYY1001$101^1.
```

This task is performed by the states `loc0`, `loc1`, ..., `loc6`.

**Fetching the new state and symbol.** Immediately following the prefix just located is a substring of length $k+1$ that encodes the new state $q'$ and new symbol $s'$. This substring is copied into the $k+1$ squares immediately to the right of the initial dollar symbol. During the copying operation, the substitutions `0` $\to$ `X`, `1` $\to$ `Y`, and `B` $\to$ `Z` are applied. In our example, the tape now reads

```
$YXX*XXXXXXY*XXYXXYY*XXZXYZX*XYXXYYX*XYYYXXY$101^1.
```

This task is performed by the states `fetch0`, `fetch1`, ..., `fetch7`.

**Printing the new symbol.** The symbol immediately to the left of the first asterisk is the code for the new symbol $s'$. $M_u$ remembers this symbol and transfers it to the tape square immediately to the right of the caret mark. In our example, the new symbol is `0`:

$YXX*XXXXXXY*XXYXXYY*XXZXYZX*XYXXYYX*XYYYXXY$101^0.

This task is performed by states `print0`, `print1`, ..., `print7`.

**Moving the tape head.** $M_u$ now looks for the first occurrence of `X` or `Y` to the left of the caret mark; this symbol determines whether the head (caret mark) should be moved left (`X`) or right (`Y`). $M_u$ now swaps the caret mark with its left or right neighbour, as appropriate. In our example, the caret mark is shifted right:

$YXX*XXXXXXY*XXYXXYY*XXZXYZX*XYXXYYX*XYYYXXY$1010^.

This task is performed by the states `move0`, `move1`, ..., `move6`.

**Tidying the tape.** The machine encoding is returned to its original condition in readiness for the following cycle. This involves applying the substitutions `X` $\to$ `0`, `Y` $\to$ `1`, and `Z` $\to$ `B` uniformly along the tape. In our example the tape now reads

$100*0000001*0010011*00B01B0*0100110*0111001$1010^.

The task is performed by states `tidy0` and `tidy1`.

This completes the description of a typical cycle of $M_u$.

**§6.3. Removing the restrictions.** To keep the universal machine relatively simple, we have restricted the class of machines that can be directly simulated to *binary* Turing machines with input alphabet $\{0, 1\}$ and tape alphabet $\{0, 1, b\!\!\!/\}$. This is no great loss, since *any* Turing machine can be transformed into an equivalent binary Turing machine by encoding each of the tape symbols by a fixed length block of binary digits. (We shall return to this point in NOTE 7.)

However, with a certain amount of extra work, it would be possible to construct a universal Turing machine that could simulate machines with *general* tape alphabet. The encoding presented here would need to be extended, each of the symbols in the tape alphabet receiving a $k'$-bit binary code for appropriately chosen $k'$. The phases of the simulation would be much as before, although the subroutines for reading and writing the tape symbol, and shifting the tape head, would be somewhat more complicated.