Lecture 40:
Another Model of Computation

Aims:

- To look at Counter Programs, which are another formal model of computation; and
- To show that Turing machines and Counter Programs are of equivalent power.

40.1. Counter Programs

- Turing machines are only one of many formal models of computation.
- Here we describe *Counter Programs*, which are another very simple formal model.
- Counter Programs or something very like them also go under the names Counter Machines, Register Machines, Minsky Machines, etc.
- A Counter Program has a finite set of variables.
- Each variable can store a natural number (i.e. a non-negative integer).
- A Counter Program is a finite *sequence* of labelled commands, the last of which is **halt**
- The other allowable commands are:
  - $x := 0$
  - $x := y + 1$
  - $x := y - 1$ (In Counter Programs, $y - 1$ is defined to be zero if y is already 0)
  - **if** $x = 0$ **goto** G (where G is the label of a command in the Program)
- The commands are executed in sequence, but branching off to the specified command when a **goto** is encountered, and terminating when the final command in the sequence (**halt**) is encountered.
- **Example 1.** A Counter Program having variables u, x and y which, if started in configuration $\langle u = u_0, x = x_0, y = y_0\rangle$, will halt in configuration $\langle u = 0, x = x_0 + y_0, y = 0\rangle$. In other words, it adds the initial contents of x and y and stores the result in x, also destroying the value that was in y. E.g. if initially x contains 3 and y contains 2, then afterwards x will contain 5 and y will contain 0.

Note the role of u.

1. $u := 0$;
2. **if** $y = 0$ **goto** 6;
3. $y := y - 1$;
4. $x := x + 1$;
5. **if** $u = 0$ **goto** 2;
6. **halt**

- Trace the program for initial configuration $\langle u = 28, x = 3, y = 2 \rangle$.
- **Example 2.** A Counter Program which, if started in configuration $\langle u = u_0, v = v_0, x = x_0, y = y_0, z = z_0 \rangle$, will halt in configuration $\langle u = 0, v = 0, x = 0, y = y_0, z = x_0 \times y_0 \rangle$. In other words, it multiplies the initial contents of x and y and stores the result in z, also destroying the value that was in x. u has the same role as before.

  Note the trick we use to copy y into v. And note that the second half of the program is effectively the adding program from Example 1.

1. $u := 0$;
2. $z := 0$;
3. **if** $x = 0$ **goto** 11;
4. $x := x - 1$;
5. $v := y + 1$;
6. $v := v - 1$;
7. **if** $v = 0$ **goto** 3;
8. $v := v - 1$;
9. $z := z + 1$;
10. **if** $u = 0$ **goto** 7;
11. **halt**

Module Home Page
Title Page
Page 4 of 9
Back
Full Screen
Close
Quit

- Just as with Turing machines, Counter Programs can be defined in many different ways that are of equivalent power:
  - E.g. restricting yourself to only two variables;
  - E.g. insisting that one register is used for the answer, and the rest are cleared by the end of the computation;
  - E.g. allowing infinitely many variables;
  - E.g. using slightly different commands.

40.2. Turing Machines and Counter Programs are of Equivalent Power

- Turing machines and Counter Programs are of equivalent power: any problem that can be solved by a Turing machine can be solved by a Counter Program, and *vice versa*.
- This is by no mean obvious: they seem like very different models of computation.
- We prove this by showing that
  - anything a Turing machine can do, a Counter Program can do — i.e., given a Turing machine, we show how to build a Counter Program that performs the same computation; and
  - anything a Counter Program can do, a Turing machine can do — i.e., given a Counter Program, we show how to build a Turing machine that performs the same computation.
- One possibly useful observation is that both models have a potentially infinite amount of memory: for Turing machines this comes in the form of its infinite tape; Counter Programs have a finite set of variables but each can hold an arbitrarily large value.

40.2.1. Using a Counter Program to Simulate a Turing Machine

- The first challenge is how to represent the contents of the tape as one or more numbers that can be stored in the variables of a Counter Program. This is possible because, although the tape is infinite, only finitely many cells are non-blank.
- Each member of Σ must be associated with a natural number (non-negative number).

- E.g. if $\Sigma = \{a, b, X, ␣\}$,

$$\begin{array}{ccc@{\qquad\qquad}ccc} a & : & 0 & b & : & 1 \\ X & : & 2 & ␣ & : & 3 \end{array}$$

- Then, if you have a sequence of characters $c_1, c_2, c_3, \ldots c_n$ and their corresponding numbers are $d_1, d_2, d_3, \ldots, d_n$, then these numbers can be combined into a single, unique number:

$$2^{d_1} \times 3^{d_2} \times 5^{d_3} \times \ldots \times p_n^{d_n}$$

  where p_i is the ith prime number. (Fundamental Theorem of Arithmetic!)

- E.g. $aabX␣bbX$ becomes $2^0 \times 3^0 \times 5^1 \times 7^2 \times 11^3 \times 13^1 \times 17^1 \times 19^2 = 26016185200$ (and no other string of characters map to the same number).

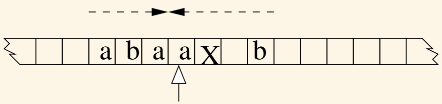
- In fact, for reasons we needn't go into, the above is a simplification. And, instead, we have to use a slight variant, such as:

$$2^{d_1+1} \times 3^{d_2+1} \times 5^{d_3+1} \times \ldots \times p_n^{d_n+1}$$

- We can use two numbers to encode
  - the cells from the leftmost non-blank up to but excluding the scanned symbol; and
  - the cells from the rightmost non-blank up to and including the scanned symbol.

  Note how the last of these is encoded 'in reverse'.

- E.g.



$$2^0 \times 3^1 \times 5^0 = 3 \qquad\qquad 2^1 \times 3^3 \times 5^2 \times 7^0 = 1350$$

- So the Counter Program has a variable x to hold the left-hand tape contents, a variable y to hold the right-hand tape contents, and a variable z to hold the Turing machine's state (plus some extra ones to help it get its work done).
- For each entry in the Turing machine's transition table, there is a sequence of Counter Program commands which alter the contents of the three variables.
- E.g. if the Turing machine moves left, we do some arithmetic on x to make it a smaller number and some arithmetic on y to make it a bigger number. (Similarly, for moving right.)
- E.g. if the Turing machine writes a symbol, then we do some arithmetic on y.
- In all cases, we also do some arithmetic on z to reflect the change of state.

40.2.2. Using a Turing Machine to Simulate a Counter Program

- Now the first challenge is how to represent the contents of the Counter Program's variables as symbols on a tape.
- Easy! Let $\Sigma = \{0, \ldots, 9, *\}$ and write out the contents of each variable onto the tape, separated by, e.g., $*$'s.
- E.g.

x	y	z
12	234	41

		$*$	1	2	$*$	2	3	4	$*$	4	1	$*$		

- In fact, contrary to what is shown in the diagram, you would probably use a binary encoding.
- For each type of command, we can work out a simple Turing machine control unit:
  - E.g. $x := 0$: scan the tape to reach the part that holds the value of x and then do some shifting (e.g. so that $*12*$ becomes $**$)
  - E.g. $x := y + 1$: involves a lot of scanning, shifting and writing.
- Then these basic machines can be combined, just like we were combining simple Turing machines into complex ones earlier.

Acknowledgements

[Har92], [LP81] and [Tru91] were all used to help me to write this lecture.

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

References

[Har92] D. Harel. *Algorithmics: The Spirit of Computing*. Addison-Wesley, 2nd edition, 1992.

[LP81] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, 1981.

[Tru91] J.K. Truss. *Discrete Mathematics for Computer Scientists*. Addison Wesley, 1991.