Notes for the Talk *What is Computation?*

Leslie Lamport

26 June 2012

Mathematical Logic

The operators $\lor$ and $\land$ are defined by:

$$\begin{array}{lllll} \text{TRUE} & \lor & \text{TRUE} & \textit{equals} & \text{TRUE} \\ \text{TRUE} & \lor & \text{FALSE} & \textit{equals} & \text{TRUE} \\ \text{FALSE} & \lor & \text{TRUE} & \textit{equals} & \text{TRUE} \\ \text{FALSE} & \lor & \text{FALSE} & \textit{equals} & \text{FALSE} \end{array}$$

and

$$\begin{array}{lllll} \text{TRUE} & \land & \text{TRUE} & \textit{equals} & \text{TRUE} \\ \text{TRUE} & \land & \text{FALSE} & \textit{equals} & \text{FALSE} \\ \text{FALSE} & \land & \text{TRUE} & \textit{equals} & \text{FALSE} \\ \text{FALSE} & \land & \text{FALSE} & \textit{equals} & \text{FALSE} \end{array}$$

These definitions imply the following equalities, for any truth value B:

$$\begin{array}{lllll} \text{TRUE} & \lor & B & \textit{equals} & \text{TRUE} \\ \text{FALSE} & \lor & B & \textit{equals} & B \\ \text{TRUE} & \land & B & \textit{equals} & B \\ \text{FALSE} & \land & B & \textit{equals} & \text{FALSE} \end{array}$$

The Binary Clock

The binary clock is described by:

$$\begin{array}{ll} Init_{clk}: & (v = 0) \lor (v = 1) \\ Next_{clk}: & \quad ((v = 0) \land (v' = 1)) \\ & \lor \ ((v = 1) \land (v' = 0)) \end{array}$$

To obtain a sequence of states that is a computation of the binary clock, we first find a value for the variable v for which $Init_{clk}$ equals TRUE. The two choices are $v = 0$ and $v = 1$. For example, substituting 0 for v in $Init_{clk}$, we have:

$$\begin{array}{llll} Init_{clk} & \textit{equals} & (0 = 0) \lor (0 = 1) & \\ & \textit{equals} & \text{TRUE} \lor \text{FALSE} & \\ & \textit{equals} & \text{TRUE} & \text{[by the definition of } \lor] \end{array}$$

Starting with the state $v = 1$, we find the next state by substituting $v = 1$ in $Next_{clk}$ to obtain

$$\begin{array}{lll} Next_{clk} \ \textit{equals} & & ((1 = 0) \land (v' = 1)) \\ & \lor & ((1 = 1) \land (v' = 0)) \\ \textit{equals} & & ((\text{FALSE}) \land (v' = 1)) \\ & \lor & ((\text{TRUE}) \land (v' = 0)) \\ \textit{equals} & & \text{FALSE} \quad [\text{because } \text{FALSE} \land B \ \textit{equals} \ \text{FALSE}] \\ & \lor & (v' = 0) \quad [\text{because } \text{TRUE} \land B \ \textit{equals} \ B] \\ \textit{equals} & v' = 0 & [\text{because } \text{FALSE} \lor B \ \textit{equals} \ B] \end{array}$$

If we substitute 1 for v in $Next_{clk}$, the only value that we can substitute for v' to make $Next_{clk}$ equal to true is 0. Therefore, from the state $v = 1$, the only possible next state is $v = 0$. So, a computation starting from the state $v = 1$ has as its next state $v = 0$. Similarly, substituting 0 for v in $Next_{clk}$, the only value we can substitute for v' that makes $Next_{clk}$ equal to TRUE is 1. Continuing this process, we see that the only computation of the binary clock starting in the state $v = 1$ is:

$$v = 1 \ \rightarrow \ v = 0 \ \rightarrow \ v = 1 \ \rightarrow \ v = 0 \ \rightarrow \ \cdots$$

Euclid's Algorithm

Euclid's algorithm computes the *greatest common divisor* of two positive integers, which is the largest positive integer that divides both of them. The algorithm is described as follows, where M and N are arbitrary fixed positive integers, and x and y are variables:

$$\begin{array}{lll} Init_{euclid} : & & (x = M) \land (y = N) \\ Next_{euclid} : & & ((x < y) \land (x' = x) \land (y' = y - x)) \\ & \lor & ((y < x) \land (y' = y) \land (x' = x - y)) \end{array}$$

A computation of this algorithm stops when the value of x equals the value of y, at which point that value equals the greatest common divisor of M and N (written $\mathbf{gcd}(M, N)$).

To see how the algorithm works, we find a computation for the case when M equals 18 and N equals 12. Finding values of x and y that make $Init_{euclid}$ true in this case yields the starting state:

$$x = 18, \ y = 12$$

To find the possible next states, we substitute 18 for x and 12 for y in $Next_{euclid}$ and solve for x' and y' as follows:

$$
\begin{array}{lll}
Next_{euclid} & \textit{equals} & \quad ((18 < 12) \land (x' = 18) \land (y' = 12 - 18)) \\
 & & \lor \; ((12 < 18) \land (y' = 12) \land (x' = 18 - 12)) \\
 & \textit{equals} & \quad (\text{FALSE} \land (x' = 18) \land (y' = 12 - 18)) \\
 & & \lor \; (\text{TRUE} \land (y' = 12) \land (x' = 18 - 12)) \\
 & \textit{equals} & \quad \text{FALSE} \\
 & & \lor \; ((y' = 12) \land (x' = 18 - 12)) \\
 & \textit{equals} & (y' = 12) \land (x' = 18 - 12) \\
 & \textit{equals} & (y' = 12) \land (x' = 6)
\end{array}
$$

This shows that the first two states of the computation are

$$x = 18,\ y = 12 \quad \to \quad x = 6,\ y = 12$$

Substituting 6 for x and 12 for y in $Next_{euclid}$ yields $x' = 6$ and $y' = 6$, so the first three states of the computation are

$$x = 18,\ y = 12 \quad \to \quad x = 6,\ y = 12 \quad \to \quad x = 6,\ y = 6$$

Substituting 6 for x and 6 for y in $Next_{euclid}$ yields

$$
\begin{array}{lll}
Next_{euclid} & \textit{equals} & \quad ((6 < 6) \land (x' = 6) \land (y' = 6 - 6)) \\
 & & \lor \; ((6 < 6) \land (y' = 6) \land (x' = 6 - 6)) \\
 & \textit{equals} & \quad (\text{FALSE} \land (x' = 6) \land (y' = 6 - 6)) \\
 & & \lor \; (\text{FALSE} \land (y' = 6) \land (x' = 6 - 6)) \\
 & \textit{equals} & \quad \text{FALSE} \\
 & & \lor \; \text{FALSE} \\
 & \textit{equals} & \text{FALSE}
\end{array}
$$

Hence, if we substitute 6 for x and 6 for y, then $Next_{euclid}$ equals FALSE no matter what values we substitute for x' and y'. This means that there is no next state from the state $x = 6$, $y = 6$, and the complete execution is

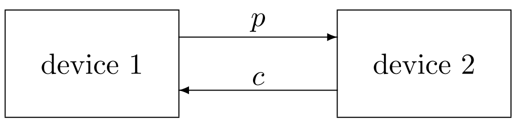
$$x = 18,\ y = 12 \quad \to \quad x = 6,\ y = 12 \quad \to \quad x = 6,\ y = 6$$

In the final state, both x and y equal 6, which equals $\mathbf{gcd}(18, 12)$.

As an exercise, calculate the computations of Euclid's algorithm for other values of M and N, such as M equal to 20 and N equal to 15.

The 2-Phase Handshake

The 2-Phase Handshake is a standard hardware signaling protocol used by two devices that alternately perform operations, the first device performing A operations and the second performing B operations. They synchronize by using two wires—one set by device 1 and read by device 2, the other set by device 2 and read by device 1.



The protocol is described using variables p and c to represent the voltages on the wires, which assume the values 0 and 1. There are also other variables that represent the states of the devices and perhaps of other wires joining them. The operations A and B performed by the two devices are represented as formulas containing these other variables (primed and unprimed). We don't care what those other variables are and what formulas A and B are. The protocol is described as follows, where the "..." stands for a formula that describes the initial values of all the variables other than p and c.

$$\begin{array}{ll} \mathit{Init}_{HS} : & (p = 0) \land (c = 0) \land \ldots \\ \mathit{Next}_{HS} : & \quad ((p = c) \land (p' = p \oplus 1) \land (c' = c) \land A) \\ & \lor \ ((p \neq c) \land (c' = c \oplus 1) \land (p' = p) \land B) \end{array}$$

where the operator $\oplus$ is defined by

$$\begin{array}{lll} 0 \oplus 0 & \mathit{equals} & 0 \\ 0 \oplus 1 & \mathit{equals} & 1 \\ 1 \oplus 0 & \mathit{equals} & 1 \\ 1 \oplus 1 & \mathit{equals} & 0 \end{array}$$

As an exercise, you can check that the following is the only computation of the 2-phase handshake, where $\stackrel{A}{\rightarrow}$ indicates a state transition in which the other variables satisfy formula A, and $\stackrel{B}{\rightarrow}$ indicates one in which they satisfy formula B.

$$\begin{array}{l} p = 0,\ c = 0 \ \stackrel{A}{\rightarrow}\ p = 1,\ c = 0 \ \stackrel{B}{\rightarrow}\ p = 1,\ c = 1 \ \stackrel{A}{\rightarrow} \\ \qquad p = 0,\ c = 1 \ \stackrel{B}{\rightarrow}\ p = 0,\ c = 0 \ \stackrel{A}{\rightarrow}\ p = 1,\ c = 0 \ \stackrel{B}{\rightarrow}\ \cdots \end{array}$$