

# Notes for *What is Computation?*

Leslie Lamport

10 December 2008

These notes accompany a talk to be given on 11 December 2008 at PARC. They will make little sense to you if you have not attended this talk.

## Mathematical Logic

The operators  $\vee$  and  $\wedge$  are defined by:

TRUE	$\vee$	TRUE	<i>equals</i>	TRUE
TRUE	$\vee$	FALSE	<i>equals</i>	TRUE
FALSE	$\vee$	TRUE	<i>equals</i>	TRUE
FALSE	$\vee$	FALSE	<i>equals</i>	FALSE

and

TRUE	$\wedge$	TRUE	<i>equals</i>	TRUE
TRUE	$\wedge$	FALSE	<i>equals</i>	FALSE
FALSE	$\wedge$	TRUE	<i>equals</i>	FALSE
FALSE	$\wedge$	FALSE	<i>equals</i>	FALSE

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

TRUE	$\vee$	$B$	<i>equals</i>	TRUE
FALSE	$\vee$	$B$	<i>equals</i>	$B$
TRUE	$\wedge$	$B$	<i>equals</i>	$B$
FALSE	$\wedge$	$B$	<i>equals</i>	FALSE

## The Binary Clock

The binary clock is described by:

$Init_{clk}$	:	$(v = 0) \vee (v = 1)$
$Next_{clk}$	:	$((v = 0) \wedge (v' = 1))$ $\vee ((v = 1) \wedge (v' = 0))$

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:

$$\begin{aligned}
 Init_{clk} & \text{ equals } (0 = 0) \vee (0 = 1) \\
 & \text{ equals } \text{TRUE} \vee \text{FALSE} \\
 & \text{ equals } \text{TRUE} \quad \quad \quad [\text{by the definition of } \vee]
 \end{aligned}$$

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

$$\begin{aligned}
 Next_{clk} & \text{ equals } ((1 = 0) \wedge (v' = 1)) \\
 & \quad \vee ((1 = 1) \wedge (v' = 0)) \\
 & \text{ equals } ((\text{FALSE}) \wedge (v' = 1)) \\
 & \quad \vee ((\text{TRUE}) \wedge (v' = 0)) \\
 & \text{ equals } \text{FALSE} \quad [\text{because } \text{FALSE} \wedge B \text{ equals } \text{FALSE}] \\
 & \quad \vee (v' = 0) \quad [\text{because } \text{TRUE} \wedge B \text{ equals } B] \\
 & \text{ equals } v' = 0 \quad [\text{because } \text{FALSE} \vee B \text{ equals } B]
 \end{aligned}$$

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:

$$v = 1 \rightarrow v = 0 \rightarrow v = 1 \rightarrow v = 0 \rightarrow \dots$$

## 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:

$$\begin{aligned}
 Init_{euclid} & : (x = M) \wedge (y = N) \\
 Next_{euclid} & : ((x < y) \wedge (x' = x) \wedge (y' = y - x)) \\
 & \quad \vee ((y < x) \wedge (y' = y) \wedge (x' = x - y))
 \end{aligned}$$

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:

$$\begin{aligned} Next_{euclid} \text{ equals } & ((18 < 12) \wedge (x' = 18) \wedge (y' = 12 - 18)) \\ & \vee ((12 < 18) \wedge (y' = 12) \wedge (x' = 18 - 12)) \\ \text{equals } & (\text{FALSE} \wedge (x' = 18) \wedge (y' = 12 - 18)) \\ & \vee (\text{TRUE} \wedge (y' = 12) \wedge (x' = 18 - 12)) \\ \text{equals } & \text{FALSE} \\ & \vee ((y' = 12) \wedge (x' = 18 - 12)) \\ \text{equals } & (y' = 12) \wedge (x' = 18 - 12) \\ \text{equals } & (y' = 12) \wedge (x' = 6) \end{aligned}$$

This shows that the first two states of the computation are

$$x = 18, y = 12 \rightarrow x = 6, y = 12$$

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

$$x = 18, y = 12 \rightarrow x = 6, y = 12 \rightarrow x = 6, y = 6$$

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

$$\begin{aligned} Next_{euclid} \text{ equals } & ((6 < 6) \wedge (x' = 6) \wedge (y' = 6 - 6)) \\ & \vee ((6 < 6) \wedge (y' = 6) \wedge (x' = 6 - 6)) \\ \text{equals } & (\text{FALSE} \wedge (x' = 6) \wedge (y' = 6 - 6)) \\ & \vee (\text{FALSE} \wedge (y' = 6) \wedge (x' = 6 - 6)) \\ \text{equals } & \text{FALSE} \\ & \vee \text{FALSE} \\ \text{equals } & \text{FALSE} \end{aligned}$$

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

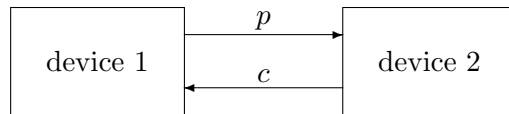
$$x = 18, y = 12 \rightarrow x = 6, y = 12 \rightarrow x = 6, y = 6$$

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  $c$ .

$$Init_{HS} : (p = 0) \wedge (c = 0) \wedge \dots$$

$$Next_{HS} : ((p = c) \wedge (p' = p \oplus 1) \wedge (c' = c) \wedge A) \\ \vee ((p \neq c) \wedge (c' = c \oplus 1) \wedge (p' = p) \wedge B)$$

where the operator  $\oplus$  is defined by

$$\begin{aligned} 0 \oplus 0 & \text{ equals } 0 \\ 0 \oplus 1 & \text{ equals } 1 \\ 1 \oplus 0 & \text{ equals } 1 \\ 1 \oplus 1 & \text{ equals } 0 \end{aligned}$$

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

$$\begin{array}{ccccccc}
 p = 0, c = 0 & \xrightarrow{A} & p = 1, c = 0 & \xrightarrow{B} & p = 1, c = 1 & \xrightarrow{A} & \\
 & & & & & & \\
 & & p = 0, c = 1 & \xrightarrow{B} & p = 0, c = 0 & \xrightarrow{A} & p = 1, c = 0 & \xrightarrow{B} & \dots
 \end{array}$$