Thinking of Classical Algorithms in a Conservative Model of Computing

> Think about computation as a physical process that evolves a prescribed initial configuration of a computing machine, called INPUT, into some final configuration, called OUTPUT. We shall refer to the configurations as states Ekert.

So to reiterate our computational process is composed of a sequence of computational steps wherein each step evolves the current state into the next state.

A state in the most simplest of cases can be thought as n bit vector.

For our purposes we can think of an n bit state vector as a contiguous binary array in memory $u[0:n]$.

Main

Two integers a and b are added together using this conservative logic adder algorithm.

In a conventional classical model of computing, we might have the two integers a and b in memory and perform the following to add:

$$a := a + b$$

That is, the result of the sum of a and b is assigned to a. However, in conservative logic, we must ensure reversibility is preserved. So we introduce a third integer c:

$$c := a + b$$

That is, the result of summing a and b is stored in c. Evidently, this requires c to be initialized to zero to ensure we properly get the sum of a and b.

Input: Three integers a, b, and c :

$$\begin{aligned} a &:= a \\ b &:= b \\ c &:= \vec{0} \end{aligned}$$

An m bit binary array binary $u[0:m]$, is initialized so that a b and c are stored in sub arrays of $u[0:m]$

- $u[0:n] := bin(a)$

- $u[n:2n] := bin(b)$
- $u[2n:3n] := 0$

Output: We add a and b, and place the result into c.

$$a := a$$
$$a := a$$
$$c := a + b$$

- $u[0:n] := bin(a)$
- $u[n:2n] := bin(b)$
- $u[2n:3n] := u[0:n] + u[n:2n]$

In other words:

$$u[0:m] := u[0:n] \otimes u[n:2n] \otimes \{u[0:n] + u[n:2n]\}$$

We define an adder algorithm which stores the result of adding a and b into c in a conservative model with the following initial and final states.

Initial State:

$$(\forall | 0 \leq i \leq n : c(i) := 0)$$

Final State:

$$(\forall | 0 \leq i \leq n : c(i) := a(i) \oplus b(i) \oplus (a(i-1) \wedge b(i-1)))$$

Or equivalently in mod 2 arithmetic

$$(\forall | 0 \leq i \leq n : c(i) := a(i) + b(i) + a(i-1)b(i-1))$$