CS31 (Algorithms), Spring 2020 : Lecture 1 Supp

Topic: Algorithms with numbers, Intro to Algorithms Analysis

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors.
Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

---

1 Correctness of the Addition Algorithm

We start with the subroutine for adding one-bit numbers. We denote this the BIT-ADD routine which takes input three bits b_1, b_2, b_3 and returns two bits (c, s). Note that the binary number with 'first' digit c and 'second' digit s is precisely $2c + s$. For instance, the number 10 is $2 \cdot 1 + 0 = 2$ and the number 11 is $2 \cdot 1 + 1 = 3$. The property of BIT-ADD is that it returns (c, s) with the property $b_1 + b_2 + b_3 = 2c + s$. This subroutine is "hard-coded" using the following truth table.

b_1	b_2	b_3	(c, s)
0	0	0	(0,0)
0	0	1	(0,1)
0	1	0	(0,1)
1	0	0	(0,1)
0	1	1	(1,0)
1	0	1	(1,0)
1	1	0	(1,0)
1	1	1	(1,1)

You should check the above table satisfies $b_1 + b_2 + b_3 = 2c + s$.

Armed with this, we can define our grade-school addition. This is slightly (more wastefully) defined below than in the lecture notes in that we are defining a "carry array". This is purely for the convenience of the proof that is about to follow.

procedure ADD($a[0:n-1], b[0:n-1]$):
    ▷ *The two numbers are a and b*
    Initialize $\mathsf{carry}[0:n] \leftarrow 0$ to all zeros.
    Initialize $c[0:n]$ to all zeros ▷ $c[0:n]$ will finally contain the sum
    for $i = 0$ to $n-1$ **do**:
        $(\mathsf{carry}[i+1], c[i]) \leftarrow$ BIT-ADD$(a[i], b[i], \mathsf{carry}[i])$
        ▷ *Invariant:* $a[i] + b[i] + \mathsf{carry}[i] = 2\mathsf{carry}[i+1] + c[i]$
    $c[n] \leftarrow \mathsf{carry}[n]$
    return c

Remark: *The above algorithm returns an $(n+1)$-bit number whose $(n+1)$th bit is 0 if the final carry is 0, otherwise it is 1. Before going into the proof of correctness, do you see why two n bit numbers cannot give a number with $> n+1$ bits?*

Theorem 1. The algorithm ADD is correct.

Proof. To prove ADD is correct, we need to show no matter what a, b is, the number represented by the bit-array $c[0:n]$ is precisely $a+b$. There is really no two ways to prove this – we look at the algorithm and see what the $c[i]$'s are and try to show that

$$\sum_{i=0}^{n} c[i] \cdot 2^i = \sum_{i=0}^{n-1} a[i] \cdot 2^i + \sum_{i=0}^{n-1} b[i] \cdot 2^i$$

To do so, we use the property of BIT-ADD stated in Line 7 of ADD:

$$\text{For all } 0 \le i \le n-1, \quad c[i] = a[i] + b[i] + (\mathsf{carry}[i] - 2\mathsf{carry}[i+1]) \tag{1}$$

Multiplying both sides by 2^i and adding, we get

$$\sum_{i=0}^{n-1} c[i] \cdot 2^i = \left(\sum_{i=0}^{n-1} a[i] \cdot 2^i\right) + \left(\sum_{i=0}^{n-1} b[i] \cdot 2^i\right) + \left(\sum_{i=0}^{n-1} \mathsf{carry}[i] \cdot 2^i - \sum_{i=0}^{n-1} \mathsf{carry}[i+1] \cdot 2^{i+1}\right)$$

We are done proving $c = a+b$. To see this, observe LHS is precisely $c - c[n] \cdot 2^n = c - \mathsf{carry}[n] \cdot 2^n$. The first parenthesized item of the RHS is a. The second parenthesized item of the RHS is b. The third is interesting; if you open up the summation you see that many terms cancel out and evaluates to $\mathsf{carry}[0] \cdot 2^0 - \mathsf{carry}[n] \cdot 2^n$ (make sure you see this.). This canceling behavior is often seen in summations and is given a name in math: it is said that this summation *telescopes* to only two terms, much like a long elongated telescope folds into one compact tube. □

Phew! Our grade school teacher was correct.

2 Subtraction

There are actually two ways to subtract binary numbers. One is just the grade-school algorithm using a "borrow" instead of a "carry". However, there is another pretty nifty way to subtract using the *method of complements*.

The algorithm is as follows. It assumes the subroutine COMPLEMENT which takes a bit-array and flips it. That is, wherever there is a 0 it makes it a 1 and vice-versa.

```
procedure SUBTRACT(a[0 : n − 1], b[0 : n − 1]):
    ▷ The two numbers are a and b; assumption a ≥ b
    a′ ← COMPLEMENT(a).
    c ← ADD(a′, b).
    return c′ ← COMPLEMENT(c).
```

Theorem 2. The algorithm SUBTRACT behaves correctly.

Proof. First, given any number n-bit number x given as a bit-array $x[0:n-1]$, we observe that $x' =$ COMPLEMENT(x) is simply the number $\left(2^{n+1}-1\right) - x$. Indeed,

$$x = \sum_{i=0}^{n} x[i] 2^i \quad \text{and} \quad x' = \sum_{i=0}^{n} (1 - x[i]) 2^i = \sum_{i=0}^{n} 2^i - x = \left(2^{n+1}-1\right) - x$$

where we use the formula for a sum of geometric series.

Next, we argue that if a and b are both n-bits *and* $a \geq b$, then $c = a' + b$ is also at most n-bits long. Indeed, $c = (2^{n+1} - 1) - (a - b)$. If $a \geq b$, then $c \leq 2^{n+1} - 1$ implying it is at most n-bits long.

Thus, COMPLEMENT(c), the number we return, is $\left(2^{n+1} - 1\right) - c = (a - b)$. Done. □

3 Correctness of the Multiplication Algorithm

In this section, we prove the correctness of the MULT algorithm by induction. This is the method many of you may have seen in CS30.

1: **procedure** MULT(x, y):
2: ▷ *The two numbers are input as bit-arrays; x has n bits, y has m bits. $n \geq m$.*
3: **if** $y = 0$ **then**: ▷ Base Case
4: **return** 0 ▷ An all zero bit-array
5: $x' \leftarrow (2x)$; $y' \leftarrow \lfloor y/2 \rfloor$
6: $z \leftarrow$MULT(x', y')
7: **if** y is even **then**:
8: **return** z
9: **else**:
10: **return** ADD(z, x)

For a pair of natural numbers (x, y) with $x \geq y$, we say MULT(x, y) works properly if it returns $x \cdot y$.

Theorem 3. MULT(x, y) works properly on all pairs of numbers x, y.

Proof. Let $P(n)$ be the predicate which is true if MULT(x, n) works properly on pairs (x, n) with $x \geq n$. Observe that if $\forall n \in \mathbb{N} : P(n)$ is true, then the theorem holds. Therefore, we proceed to prove $\forall n \in \mathbb{N} : P(n)$ is true by inducton.

Base Case: $n = 1$. We need to show that MULT$(x, 1)$ behaves properly for all $x \geq 1$. That is, we need to show MULT$(x, 1)$ returns $x \cdot 1 = x$. Indeed, the algorithm runs Line 4 in this case and returns x. So $P(1)$ is true.

Inductive Case: Fix a natural number $k \geq 1$. Assume $P(1), P(2), \cdots, P(k)$ is true. We need to show $P(k+1)$ is true. That is, we need to show for any number $x \geq k+1$, MULT$(x, k+1)$ returns $x \cdot (k+1)$. To that end, *fix* a number $x \geq k+1$.

Let us consider the behavior of the algorithm. In Line 5, we set $y' = \lfloor (k+1)/2 \rfloor$. Since $k \geq 1$, $(k+1) \geq 2$, we have $y' \geq 1$. Furthermore, $y' \leq k$. This is because $k \geq 1$ implies $2k \geq k+1$ which in turn implies $k \geq (k+1)/2 \geq y'$. In sum, $1 \leq y' \leq k$.

Since $P(y')$ is true by the Induction Hypothesis, MULT(x', y') returns $x' \cdot y'$. Thus, the z set in Line 6 is indeed $z = x' \cdot y' = 2x \cdot y'$.

Now we have a simple case analysis: if $(k+1)$ is even, then $y' = (k+1)/2$, and thus $z = 2x \cdot (k+1)/2 = x \cdot (k+1)$. Note that in the case $(k+1)$ is even, the algorithm runs Line 8 and returns $z = x(k+1)$. Thus, in this case, $P(k+1)$ is true.

If $(k+1)$ is odd, then $y' = \lfloor (k+1)/2 \rfloor = k/2$. Thus, $z = 2x \cdot y' = xk$. Note that in the case $(k+1)$ is odd, the algorithm runs Line 10, and returns $z + x = kx + x = x(k+1)$. Thus, even in this case, $P(k+1)$ is true.

Thus, in all cases $P(k+1)$ is true. Therefore, by induction, $\forall n : P(n)$ is true.

$\square$