1 Correctness of the Addition Algorithm

We start with the subroutine for adding one-bit numbers. We denote this the \texttt{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 \texttt{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.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(0,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(0,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(1,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(1,1)</td>
</tr>
</tbody>
</table>

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.

1: \textbf{procedure} ADD\((a[0 : n - 1], b[0 : n - 1]):\)
2: \hspace{1em} \triangleright \text{The two numbers are } a \text{ and } b
3: \hspace{1em} Initialize \(\text{carry}[0 : n] \leftarrow 0\) to all zeros.
4: \hspace{1em} Initialize \(c[0 : n]\) to all zeros \(\triangleright c[0 : n] \text{ will finally contain the sum}\)
5: \hspace{1em} \textbf{for } i = 0 \text{ to } n - 1 \textbf{ do:}
6: \hspace{2em} \triangleright \text{Invariant: } a[i] + b[i] + \text{carry}[i] = 2 \cdot \text{carry}[i+1] + c[i]
7: \hspace{2em} (\text{carry}[i+1], c[i]) \leftarrow \text{BIT-ADD}(a[i], b[i], \text{carry}[i])
8: \hspace{1em} c[n] \leftarrow \text{carry}[n]
9: \hspace{1em} \textbf{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:

For all \(0 \leq i \leq n - 1\),

\[
c[i] = a[i] + b[i] + (\text{carry}[i] - 2\text{carry}[i+1])
\]

(1)

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

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

We are done proving \(c = a + b\). To see this, observe LHS is precisely \(c - c[n] \cdot 2^n = c - \text{carry}[n] \cdot 2^n\). 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 \(\text{carry}[0] \cdot 2^0 - \text{carry}[n] \cdot 2^n\) (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.