CSC165H, Mathematical expression and reasoning for computer science week 8

20th July 2005

Gary Baumgartner and Danny Heap
heap@cs.toronto.edu
SF4306A
416-978-5899
http://www.cs.toronto.edu/~heap/165/S2005/index.shtml

Proof structure summary on-line

I've put a link to Gary Baumgartner's notes on proof structure on-line, on the assignments page. Please pay particular attention to the recursive expansion of $A \Leftrightarrow B$. This structure for proving bi-implication is probably easier and less error-prone than the symmetrical approach we discussed in lecture. I will instruct the marker for A3 to accept either.

Binary (base 2) notation

In our everyday life we write numbers in decimal (base 10) notation (although I heard of one kid who learned to use the fingers of her left hand to count from 0 to 31 in base 2). In decimal the sequence of digits 20395 represents (parsing from the right:

$$5 + 9(10) + 3(100) + 0(1000) + 2(10000) \quad =$$

$$5(10^0) + 9(10^1) + 3(10^2) + 0(10^3) + 2(10^4)$$

Each position represents a power of 10, and 10 is called the BASE. Each position has a digit from $[0, 9]$ representing how many of that power to add. Why do we use 10? Perhaps due to having 10 fingers (however, humans at various times have used base 60, base 20, and mixed base 20,18 (Mayans)). In the last case there were $(105)_{20,18}$ days in the year. Any integer with absolute value greater than 1 will work (so experiment with base -2).

Consider using 2 as the base for our notation. What digits should we use?[1] We don't need digits 2 or higher, since they are expressed by choosing a different position for our digits (just as in base 10, where there is no single digit for numbers 10 and greater).

Here are some examples of binary numbers:

$$(10011)_2$$

represents

$$1(2^0) + 1(2^1) + 0(2^2) + 0(2^3) + 1(2^4) = (19)_{10}$$

We can extend the idea, and imitate the decimal point (with a "binary point"?) from base 10:

$$(1011.101)_2 = 19\frac{5}{8}$$

How did we do that?[2] Here are some questions:

- How do you multiply two base 10 numbers?[3] Work out 37×43.
- How do you multiply two binary numbers?[4]
- What does "right shifting" (eliminating the right-most digit) do in base 10?[5]
- What does "right shifting" do in binary?[6]
- What does the rightmost digit tell us in base 10? In binary?

Convert some numbers from decimal to binary notation. Try 57. We'd like to represent 57 by adding either 0 or 1 of each power of 2 that is no greater than 57. $57 = 32 + 16 + 8 + 1 = (111001)_2$. We can also fill in the binary digits, systematically, from the bottom up, using the % operator (the remainder after division, at least for positive arguments)

$$\begin{aligned}
57\%2 = 1 &\quad \textit{so} \quad (????1)_2 \\
(57-1)/2 = 28\%2 = 0 &\quad \textit{so} \quad (????01)_2 \\
28/2 = 14\%2 = 0 &\quad \textit{so} \quad (???001)_2 \\
14/2 = 7\%2 = 1 &\quad \textit{so} \quad (??1001)_2 \\
(7-1)/2 = 3\%2 = 1 &\quad \textit{so} \quad (?11001)_2 \\
\\
(3-1)/2 = 1\%2 = 1 &\quad \textit{so} \quad (111001)_2
\end{aligned}$$

Addition in binary is the same as (only different from...) addition in decimal. Just remember that $(1)_2 + (1)_2 = (10)_2$. If we add two binary numbers, this tells us when to "carry" 1:

$$\begin{array}{rl}
 & 1011 \\
+ & 1011 \\
 & - - - \\
 & 10110
\end{array}$$

LOG$_2$

How many 5-digit binary numbers are there (including those with leading 0s)? These numbers run from $(00000)_2$ through $(11111)_2$, or 0 through 31 in decimal — 32 numbers. Another way to count them is to consider that there are two choices for each digit, hence 2^5 strings of digits. If we add one more digit we get twice as many numbers. Every digit doubles the range of numbers, so there are two 1-digit binary numbers (0 and 1), four 2-digit binary numbers (0 through 3), 8 3-digit binary numbers (0 through 7), and so on.

Reverse the question: how many digits are required to represent a given number. In other words, what is the smallest integer power of 2 needed to exceed a given number? LOG$_2 x$ is the power of 2 that gives $2^{\log_2 x} = x$. You can think of it as how many times you must multiply 1 by 2 to get x, or roughly the number of digits in the binary representation of x. (the precise number of digits needed is $\lfloor (log_2 x) + 1 \rfloor$, which is equal to (why?) $\lfloor log_2 x \rfloor + 1$).

Loop invariant for base 2 multiplication

We currently represent integers in binary on a computer, since it is easy to multiply or divide by 2 (left or right shift), and to determine even/odd. Here's a multiplication algorithm that uses these fast operations (see MULT(m,n) in Example.java on our web page).

We will show that any iteration of the loop preserves the claimed relationship between the variables, that is if $z = mn - xy$ before executing the loop and $x \neq 0$, then after executing the loop the loop body, $z = mn - xy$ is still true. Here's a sketch of a proof:

> Let $x', y', z', x'', y'', z'', m, n \in \mathbf{Z}$, assume the $'$ elements related to the $''$ elements by the action of the loop. Observe that the values of m and n are never changed in the loop.
>
> Assume $z' = mn - x'y'$
>
> Case 1, x' odd.
>
> Then $z'' = z' + y', x'' = (x' - 1)/2.0, y'' = 2y'$
>
> So
>
> $$\begin{aligned} mn - x''y'' &= mn - (x' - 1)/2.0 * 2y' - \text{–x odd} \\ &= mn - x'y' + y' \\ &= z' + y' \\ &= z'' \end{aligned}$$
>
> Case 2, x' even
>
> Then
>
> $$z'' = z', x'' = x'/2.0, y'' = 2y'$$
>
> So
>
> $$\begin{aligned} mn - x''y'' &= mn - x'/2.0 * 2y' \\ &= mn - x'y' \\ &= z' \\ &= z'' \end{aligned}$$
>
> Since x is either even or odd, in all cases $mn - x''y'' = z''$
>
> Thus $mn - x'y' = z' \Rightarrow mn - x''y'' = z''$.
>
> Since $x', x'', y', y'', z', z'', m, n$ are arbitrary elements of $\mathbf{Z}$, $\forall x', x'', y', y'', z', z'', m, n \in \mathbf{Z}$ $mn - x'y' = z'$ implies $mn - x''y'' = z''$.

Run time and constant factors

When calculating the running time of a program, we may know how many basic "steps" it takes as a function of input size, but we may not know how long each step takes on a particular computer. We would like to estimate the running time while ignoring constant factors. So, for example $t(n) = 3n^2$, $t(n) = 8n^2$, and $t(n) = n^2/2$ are considered the same ignoring ("to within") constant factors. This means that lower order terms can be ignored as well, so $f(n) = 3n^2$ and $g(n) = 3n^2 + 2$ are considered "the same" as are $h(n) = 3n^2 + 2n$ and $j(n) = 5n^2$. Notice that

$$\forall n \in \mathbf{N}, n \geq 1 \Rightarrow f(n) \leq g(n) \leq h(n) \leq j(n)$$

but, there's always a constant factor that can reverse any of these inequalities.

“Sufficiently large” n

Big O

Notes

[1]From 0 to $(2-1)$, if we work in analogy with base 10.

[2]To parse the 0.101 part, calculate $0.101 = 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3})$.

[3]You should be able to look up this algorithm in an elementary school textbook.

[4]Same as the previous exercise, but only write numbers that have 0's and 1's, and do binary addition.

[5]Integer divides by 10.

[6]Integer divide by 2.