Synthesis Exemplar - CS 5800
(fancy word for example)
Last revision: October 2021
Dr. Alan Jamieson

Author note: this is just an example of the kind of thing I'm looking for with each of the sections of the synthesis assignment. Feel free to model each of your sections based on this example! Especially note that this example is pretty short, others will be much longer. You don't need a novel to explain a topic, but it is on you to describe it with sufficient detail (with examples if necessary) to prove to me that your understanding is solid. --A

2 Divide and Conquer

…

2.2 Divide and Conquer Multiplication

We can speed up integer multiplication by using a divide and conquer technique with the binary representation of the numbers we are trying to multiply. The algorithm relies on a formula:

$$\begin{aligned} x*y &= (2^{n/2}\,XL + XR)(2^{n/2}\,YL + YR) \\ &= 2^{n}XL*YL + 2^{n/2}(XL*YR + XR*YL) + XR*YR \end{aligned}$$

Where x is split into two halves XL and XR (left half and right half, respectively), and y is split into two halves YL and YR.

This algorithm gets us a time function of $T(n) = 4T(n/2) + O(n)$. Each of the 4 multiplications (XL*YL, XL*YR, XR*YL, XR*YR) are on a binary string that is half the size of the original, thus n/2. O(n) is part of the analysis because it encompasses all of the additions needed to combine the divided parts into the solution.

However, we can further speed this up by using a trick from Gauss. Gauss noted that when you multiply two complex numbers:

$(a + bi) * (c + di) = ac - bd + (bc + ad)i$

You can reduce the number of multiplications we use by expanding the (bc + ad) term. This results in a new formula:

$(a + bi) * (c + di) = ac - bd + ((a+b) * (c+d) - ac - bd)i$

How does Gauss's trick help us? Here's a partial example. Given two binary integers:

X = 11001000
Y = 10110110

We split X and Y each into bit strings half the length.

XL = 1100
XR = 1000
YL = 1011
YR = 0110

We can use Gauss's trick to change our formula slightly by expanding the (XL*YR + XR * YL) part of our formula:

$$\begin{aligned} xy &= 2^n XL*YL + 2^{n/2}(XL * YR + XR * YL) + XR*YR \\ &= 2^n XL*YL + 2^{n/2}((XL + XR)(YL + YR) - XL*YL - XR*YR) + XR*YR \end{aligned}$$

Saving us one recursive call! This uses 3 total multiplications (XL*YL, XR*YR, (XL+XR)*(YL+YR)), but note that we need to use some storage to store the values we need to use more than once.

Note, each of the multiplications here (e.g. XL * YL) is an instance of this multiplication on a bit string half the size of the original. While we don't go through this in this example, this will recursively call our multiplication routine over and over again, each time slicing our bit string by half.

The pseudocode algorithm, as given in class:

```
function multiply(x,y)
Input: n-bit positive integers x and y
Output: Their product

if n = 1: return x*y

XL, XR = leftmost ceiling(n/2), rightmost floor(n/2) bits of x
YL, YR = leftmost ceiling(n/2), rightmost floor(n/2) bits of y

P1 = multiply(XL, YL)
P2 = multiply(XR, YR)
P3 = multiply(XL + XR, YL + YR)

return P1 * 2^n + (P3 - P1 - P2) * 2^floor(n/2) + P2
```

Exercise: Given the following binary numbers, utilize the divide and conquer multiplication algorithm to multiply them together:

X = 01100100
Y = 01001001

Solution:

X * Y:

First split X and Y into their respective halves:

XL = 0110
XR = 0100
YL = 0100
YR = 1001

Then calculate:

A = XL * YL
B = XR * YR
C = (XL + XR) * (YL + YR) = 1010 * 1101

Next iteration:

XL * YL (0110 * 0100):

Split XL and YL into respective halves:
XLL = 01
XLR = 10
YLL = 01
YLR = 00

Calculate:
A = XLL * YLL
B = XLR * YLR
C = (XLL + XLR) * (YLL + YLR) = 11 * 01

XR * YR (0100 * 1001):

Split XR and YR into respective halves:
XRL = 01
XRR = 00
YRL = 10
YRR = 01

Calculate:
A = XRL * YRL
B = XRR * YRR

C = (XRL + XRR) * (YRL + YRR) = 01 * 11

1010 * 1101:

Finally, split T = 1010 * Q = 1101 into respective halves
TL = 10
TR = 10
QL = 11
QR = 01

Calculate:
A = TL * QL
B = TR * QR
C = (TL + TR) * (QL + QR) = 100 * 100

Final iteration (note that splits here give us our base case of n = 1):

XLL * YLL (01 * 01):
Split:
XLLL = 0
YLLL = 0
XLLR = 1
YLLR = 1

Calculate:
A = XLLL * YLLL = 0 * 0 = 0
B = XLLR * YLLR = 1 * 1 = 1
C = (XLLL + XLLR) * (YLLL * YLLR) = 1 * 1 = 1

XLR * YLR (10 * 00):
Split:
XLRL = 1
YLRL = 0
XLRR = 0
YLRR = 0

Calculate:
A = XLRL * YLRL = 1 * 0 = 0
B = XLRR * YLRR = 0 * 0 = 0
C = (XLRL + XLRR) * (YLRL + YLRR) = 1 * 0 = 0

11 * 01:
Split:

1L = 1
2L = 0
1R = 1
2R = 1

Calculate:
A = 1L * 2L = 1 * 0 = 0
B = 1R * 2R = 1 * 1 = 1
C = (1L + 1R) * (2L + 2R) = (1 + 1) * (0 + 1) = 10

XRL * YRL (01 * 10):
Split:
XRLL = 0
YRLL = 1
XRLR = 1
YRLR = 0

Calculate:
A = XRLL * YRLL = 0 * 1 = 0
B = XRLR * YRLR = 1 * 0 = 0
C = (XRLL + XRLR) * (YRLL + YRLR) = (0 + 1) * (1 + 0) = 1

XRR * YRR (00 * 01):
Split:
XRRL = 0
YRRL = 0
XRRR = 0
YRRR = 1

Calculate:
A = XRRL * YRRL = 0 * 0 = 0
B = XRRR * YRRR = 0 * 1 = 0
C = (XRRL + XRRR) * (YRRL + YRRR) = (0 + 0) * (0 + 1) = 0

01 * 11:
Split:
1L = 0
2L = 1
1R = 1
2R = 1

Calculate:
A = 1L * 2L = 0 * 1 = 0
B = 1R * 2R = 1 * 1 = 1

C = (1L + 1R) * (2L + 2R) = (0 + 1) * (1 + 1) = 10

TL * QL (10 * 11):
Split:
TLL = 1
QLL = 1
TLR = 0
QLR = 1

Calculate:
A = TLL * QLL = 1 * 1 = 1
B = TLR * QLR = 0 * 1 = 0
C = (TLL + TLR) * (QLL + QLR) = (1 + 0) * (1 + 1) = 10

TR * QR (10 * 01):
Split:
TRL = 1
QRL = 0
TRR = 0
QRR = 1

Calculate:
A = TRL * QRL = 1 * 0 = 0
B = TRR * QRR = 0 * 1 = 0
C = (TRL + TRR) * (QRL + QRR) = 1 * 1 = 1

100 * 100:

As a note, this would normally also get split again into a 10 and 0 for both bit strings. For the purposes of this solution, we take a small shortcut and go ahead and note that this multiplies to 10000 (16).

Now we can start to put things back together, working backwards from the recursion:

XLL * YLL = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 0 + 1
= 1

XLR * YLR = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 0 + 0
= 0

11 * 01 = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 1 * 2 + 1

= 11

XRL * YRL = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 1 * 2 + 0
= 10

XRR * YRR = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 0 + 0
= 0

01 * 11 = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 1 * 2 + 1
= 11

TL * QL = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 1*2^2 + 1 * 2 + 0
= 110

TR * QR = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 0 + 1*2 + 0
= 10

100 * 100 = 10000

XR * YR = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 10 * 2^4 + 1 * 2^2 + 0
= 100100

XL * YL = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 1 * 2^4 + 10 * 2^2 + 0
= 11000

1010 * 1101 = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 110 * 2^4 + 1000 * 2^2 + 10
= 10000010

X * Y = A * 2^n + (C - B - A) * 2^floor(n/2) + B
= 11000 * 2^8 + 1000110 * 2^4 + 100100
= 1110010000100

Additional note: this is the kind of complexity I’m looking for in the exercise! A few simpler, a few more complex, you’re good. --A