CS/Math 240: Introduction to Discrete Mathematics 3/3/2011

Lecture 13 : Program Analysis

Instructor: Dieter van Melkebeek Scribe: Dalibor Zelený

DRAFT

Last time we started discussing program analysis. We used recurrence relations to analyze the running times of various, mostly recursive, programs. We continue with this today.

13.1 Euclid's Algorithm

Today we analyze Euclid's algorithm for finding the greatest common divisor of two integers. You proved correctness of the iterative version of this algorithm on the fourth homework. Today we focus on the recursive version of this algorithm. As you can see below, the recursive description is much more succinct than the iterative description.

```
Algorithm Euclid($a, b$)
    Input: $a, b \in \mathbb{N}$, $a \leq b$, $b \neq 0$
    Output: $\gcd(a, b)$
(1) if $a = 0$ then return $b$
(2) $r \leftarrow b - \lfloor b/a \rfloor \cdot a$
(3) return Euclid($r, a$)
```

We first argue that the recursive algorithm `Euclid` correctly finds greatest common divisors, and then analyze its running time.

13.1.1 Correctness of the Algorithm

For partial correctness, we assume as our induction hypothesis that all recursive calls whose inputs satisfy the preconditions produce the correct output, and show that this implies the algorithm is going to return the correct result as well, provided it terminates. There are two cases to consider.

Case 1: $a = 0$. The algorithm returns b in this case, which is the correct answer because every integer divides zero, and b is the largest divisor of b.

Case 2: $a \neq 0$. The algorithm calls `Euclid(`r, a`)` where r is the remainder of b after division by a, so $0 \leq r < a$. Because $0 \leq r < a$, the input (r, a) satisfies the preconditions of `Euclid`, and `Euclid(`r, a`)` returns $\gcd(r, a)$ by the induction hypothesis. We also know by a result you proved on the fourth homework that $\gcd(r, a) = \gcd(a, b)$, so the recursive call returns $\gcd(a, b)$ as desired.

For termination, observe that in each recursive call, the value of the first argument to `Euclid` decreases because on input (a, b), the recursive call is with input (r, a) and $r < a$. Since the first argument to `Euclid` is a non-negative integer, it becomes zero after at most a recursive calls, and the algorithm halts at that point.

13.1.2 Analysis of the Algorithm

Now that we know the algorithm works, let's analyze its running time. Our measure of complexity is the number of recursive calls on input (a, b), which we call $R(a, b)$. By an argument from the proof of correctness, we have $R(a, b) = 1 + R(r, a)$ where r is the remainder of b after dividing by a.

We have an upper bound $R(a, b) \leq a$ from the proof of termination, which roughly matches the bound $R(a, b) \leq \max(a, b)$ we found for the version of the algorithm from last time. As we see today, `Euclid` actually performs much better than the algorithm from last time.

Notice that $R(a, b)$ is different for different $a, b \leq n$. For example, $R(0, 5) = 0$, but $R(2, 5) = 2$. This makes reasoning about $R(a, b)$ difficult, so we define some quantities in terms of the maximum of a and b (call this maximum n) instead. The quantity $R'(n)$ is similar to the one we used in the analysis of last lecture's algorithm for finding greatest common divisors. It is the largest number of recursive calls made by `Euclid` on input (a, b) with $a, b \leq n$. Today's expression for $R'(n)$ is simpler because we know which of the two inputs is larger.

$$R'(n) = \max\left(\{R(a, b) \mid a, b \in \mathbb{N}, a \leq b, b \neq 0, b \leq n\}\right)$$

Unfortunately, finding an expression even for $R'(n)$ becomes difficult with this algorithm, so we turn things around and ask what is the smallest value of n for which we can get an input (a, b), with $a, b \leq n$, on which `Euclid` makes k recursive calls. To that end, we define $N(k)$ below.

$$N(k) = \min\big(\{n \in \mathbb{N} \mid R'(n) \geq k\}\big)$$

Let's compute $N(k)$ for some values of k to get a sense of how the quantity changes with k.

We have $N(0) = 1$. An example of an input (a, b) with $a, b \leq 1$ on which `Euclid` makes no recursive calls is $(0, 1)$. On this input, we have $a = 0$, so the algorithm returns right away without making any recursive calls. Note that $N(0)$ cannot be zero because $(0, 0)$ is not a valid input to the algorithm.

Next, $N(1) = 1$. Consider the input $(1, 1)$. On this input, `Euclid` makes a recursive call with $(0, 1)$ as the input, and that call returns right away. We cannot have $N(1) = 0$ for the same reason why $N(0) \neq 0$.

We have $N(2) = 3$. For example, on input $(2, 3)$, the inputs to subsequent recursive calls are $(1, 2)$ and $(0, 1)$. We now justify that $N(2) \neq 2$. On each of the inputs $(0, 2), (1, 2), (2, 2)$, we make at most one recursive call. On input $(0, 2)$, the algorithm returns right away. On input $(1, 2)$, the only recursive call is with input $(0, 1)$, and on input $(2, 2)$, the only recursive call is with input $(0, 2)$. Finally, we have discussed all the inputs with $b \leq 1$, and we saw that on those inputs there is at most one recursive call.

We now claim $N(3) \leq 5$. This is because on input $(3, 5)$, the first recursive call is with input $(2, 3)$ and we saw that it took two more recursive calls to find $\gcd(2, 3)$. One can actually show that $N(3) = 5$. For this, we'd have to argue about all inputs with $b \leq 4$. We already argued this for all inputs with $b \leq 2$, so we know $R'(1), R'(2) < 3$. We also argued about the input $(2, 3)$. We'd still need to argue about the inputs $(0, 3)$, $(1, 3)$, $(3, 3)$, $(0, 4)$, $(1, 4)$, $(2, 4)$, $(3, 4)$, and $(4, 4)$ to find that $R'(3) = R'(4) = 2$. We leave the verification of this fact to the reader.

We also have $N(4) = 8$. We don't make the argument now and present a more general argument in a moment. For now, just observe that if we call `Euclid` with input $(5, 8)$, then the first recursive call is with input $(3, 5)$, and we know it takes three more recursive calls to get the answer on that input.

We discussed the values of $N(k)$ for a few first values of k. Now let's argue in more generality. Suppose that `Euclid` takes k recursive calls on input (a, b) and that $k \geq 1$. In this case $a \neq 0$, and the algorithm recursively calls `Euclid(`r, a`)` where $r = b - \lfloor b/a \rfloor \cdot a$ is the remainder of b after dividing by a. The call `Euclid(`r, a`)` takes $k-1$ recursive calls because $k = R(a, b) = 1 + R(r, a)$. Thus, $a \geq N(k-1)$ because $N(k-1)$ is the smallest value of the second argument to `Euclid` on which it makes $k-1$ recursive calls.

By the same token, $r \neq 0$ if $k \geq 2$. On input (r, a), the second argument of the next recursive call will be r. From that point on, it takes $k-2$ recursive calls to get the answer, so $r \geq N(k-2)$. Now $b = \lfloor b/a \rfloor \cdot a + r$, and we know $r \geq N(k-2)$, $a \geq N(k-1)$, and $\lfloor b/a \rfloor \geq 1$. Therefore, $b \geq N(k-1) + N(k-2)$, so we've shown that

$$N(k) \geq N(k-1) + N(k-2). \tag{13.1}$$

Observe that our claim that $N(4) = 8 = N(3) + N(2)$ is consistent with (13.1).

We now show that the inequality (13.1) is actually an equality. To turn (13.1) into an equality, we need to show an input with $a, b \leq N(k-1) + N(k-2)$ on which `Euclid` makes k recursive calls. For example, when $k = 4$, $(5, 8)$ is one such input.

Claim 13.1. *$N(k) = N(k-1) + N(k-2)$ for $k \geq 4$, and $(a, b) = (N(k-1), N(k))$ realizes $N(k)$ for $k \geq 3$.*

By "(a, b) realizes $N(k)$" we mean that `Euclid` makes k recursive calls on input (a, b). Also recall that $N(0) = N(1) = 1$, $N(2) = 3$, and $N(3) = 5$. A last remark before the proof is that $N(k) > 0$ for all $k \in \mathbb{N}$ ad that $N(k) \leq N(k+1)$ for all $k \in \mathbb{N}$.

Proof of Claim 13.1. We prove the claim by induction on k.

For the base case we consider $k = 3$. We have $N(k-1) = N(2) = 3$ and $N(k) = N(3) = 5$. Recall that the input $(N(k-1), N(k)) = (3, 5)$ indeed requires $k = 3$ recursive calls.

We don't need to prove the other part of Claim 13.1 for $k = 3$ because that part of the claim only mentions values of k that are at least 4. Thus, the base case is proved.

For the induction step, let $k \geq 3$, and consider the input $(a, b) = (N(k), N(k) + N(k-1))$. If we give (a, b) to `Euclid` as input, it makes a recursive call with input (r, a) where $r = b - \lfloor b/a \rfloor \cdot a$. Note that $\lfloor \frac{N(k)+N(k-1)}{N(k)} \rfloor = \lfloor \frac{N(k)}{N(k)} + \frac{N(k-1)}{N(k)} \rfloor = 1$ because $N(k)/N(k) = 1$ and $N(k-1) < N(k)$. To see the latter inequality, recall $N(k) = N(k-1) + N(k-2)$ by the induction hypothesis, and $N(k-2) \geq 1$. Hence, $r = b - a = N(k-1)$.

It follows that `Euclid(`a, b`)` makes a call to `Euclid(`$N(k-1), N(k)$`)`. Our induction hypothesis says that the call to `Euclid` on input $(N(k-1), N(k))$ takes k recursive calls, which means the call to `Euclid` on input $(a, b) = (N(k), N(k) + N(k-1))$ takes $k+1$ recursive calls. This shows that $N(k+1) \leq N(k) + N(k-1)$.

Earlier, we argued inequality (13.1) which says that $N(k+1) \geq N(k) + N(k-1)$. It follows that $N(k+1) = N(k) + N(k-1)$, and our input (a, b) that realizes $N(k+1)$ is actually $(N(k), N(k+1))$. Hence, the proof is complete. □

Finding the statement of Claim 13.1 is a result of our examination of the first few values of $N(k)$ and our subsequent analysis that led us to (13.1). Again we point out that it takes some ingenuity and playing with the problem to come up with this.

Claim 13.1 gives us the following recurrence for $N(k)$: $N(0) = N(1) = 1$, $N(2) = 3$, $N(3) = 5$, and $N(k) = N(k-1) + N(k-2)$ for $k \geq 4$. This looks like the recurrence for the Fibonacci sequence. We compare the two in Figure 13.1.

We see from Table 13.1 that $N(k) = F_{k+2}$ for $k \geq 3$.

k	0	1	2	3	4	5	6	7
$N(k)$	1	1	3	5	8	13	21	34
F_k	n/a	1	1	2	3	5	8	13
F_{k+2}	1	2	3	5	8	13	21	34

Figure 13.1: Comparing $N(k)$ with the Fibonacci sequence.

Recall that our goal is to find the complexity of the algorithm `Euclid`. So far we have only found a recurrence for a related quantity, $N(k)$. Let's start by finding an expression for $N(k)$ that is not a recurrence and then translate that to an expression for $R'(n)$.

Recall from an earlier lecture that

$$F_n = \frac{1}{\sqrt{5}}\left(\varrho^n - (1-\varrho)^n\right), \quad \varrho = \frac{1+\sqrt{5}}{2}.$$

For large n, F_n is "roughly" ϱ^n (we will discuss what we mean by "roughly" next time), so $N(k)$ is roughly ϱ^{k+2}. It follows that `Euclid` makes at most $k+2$ recursive calls on input (a, b) with $a \leq b \leq \varrho^{k+2}$. So pick $b = \varrho^{k+2}$. Then $\log_\varrho(b)$ is roughly K, and the number of recursive calls made by `Euclid(`a, b`)` is, therefore, roughly at most $\log_\varrho b$. This is a great improvement over last time's analysis where the number of recursive calls was bounded by the value of the larger input as opposed to the logarithm of that value.