CIS 575. Introduction to Algorithm Analysis
Material for January 17, 2024

Algorithms are for Implementing Specifications

1 Specifications

As computer scientists, it should be our aim that the software we develop meets the desires of the users (which may include us). To do so, it is important to state explicitly, in a more or less formal contract, what a given piece of software is supposed to do. In its most basic form, a contract will say that

1. provided that the input satisfies a given *Precondition*
2. the output will satisfy a given *Postcondition*.

Observe that item 1 states the obligation of the user, whereas item 2 states the obligation of the software developer (and if the user does not comply with the Precondition then the developer is not obliged to comply with the Postcondition).

For larger pieces of software, the input may be provided continuously and the output thus produced continuously, and one would also need to address issues such as security. But in this course, we shall focus on the basic task of transforming a given input into output that fulfills a given contract.

2 Algorithms

The task of an algorithm is to implement a given specification. An algorithm may be described using any programming language; in this course, we shall use pseudocode that (with a few exceptions to be explained along the way) should be understandable by anyone who knows at least one standard high-level programming language. We shall not rely on library functions except for the most basic ones, and shall restrict our attention to sequential (that is non-parallel) algorithms.

In addition to specifying the desired relation between input and output, a contract may require efficient use of resources, in particular short response times. Accordingly, a key focus of this course will be to analyze how much time and space a given algorithm uses, and to learn techniques for developing algorithms that have acceptable costs.

Before embarking on writing an algorithm for a given specification, it is important to ensure that the specification is indeed what the user wants. For that purpose, some kind of interaction is usually needed, as we shall now illustrate by a simple example.

3 An Example of Specification Development

Supposed we are given an informal task:

> Return in variable `y` the integer square root of the integer `x` given as input.

We would like to transform that into a mathematical specification, but may not be sure what the user means by "integer square root". Inquiring the user, we may learn that we should aim at

$$y^2 = x \tag{1}$$

but we need to raise several issues. For example, what if x is negative? If we do not want to involve complex numbers, we need the user to promise not to provide negative numbers as input, and therefore include $x \geq 0$ in the precondition.

Another issue: for positive x, equation (1) has *two* solutions. The user will probably tell us that the positive one is desired, so we add $y \geq 0$ to the postcondition.

Finally, and most critically: what if x is *not* a perfect square? Then we'll have to settle for an approximation, and may therefore relax equation (1) into

$$y^2 \leq x$$

Still, this is way too liberal: no matter which non-negative x the user provides, we can fulfill the contract by returning $y = 0$ which is surely not what the user had in mind. We therefore need to specify that y is as large as possible in that $y + 1$ will be too large, and demand $(y + 1)^2 > x$.

We are now almost done. We have constructed the postcondition

$$y \geq 0 \;\wedge\; y^2 \leq x \;\wedge\; (y + 1)^2 > x$$

but may observe that the first conjunct is redundant since it follows from the other two: if $y^2 < (y + 1)^2$ then y cannot be negative.

Collecting the pieces, we have arrived at the contract

Precondition $x \geq 0$

Postcondition $y^2 \leq x \;\wedge\; (y + 1)^2 > x$

Observe that $y = 3$ will be returned for $x = 9$, $x = 10$, and even $x = 15$. In the latter case, the user may prefer $y = 4$ (food for thought: how to modify the contract to ensure that?)