CSC236, Francois Pitt
Introduction to the Theory of Computation

Fall 2023

Contents

1 Induction

Example proof of QRT:
Theorem: $\forall m, \forall n > 0, \exists q, \exists r < n, m = qn + r$
Let $m_0 \in \mathbb{N}, n_0 \in \mathbb{N}$ and $n_0 > 0$.
Remark: we will apply well-ordering principle.
Consider $R = \{r \in \mathbb{N} : \exists q, m_0 = qn_0 + r\}$
Claim: $R \neq \emptyset$ to apply WOP.
Justification: $m_0 \in \mathbb{R} \implies m_0 = 0 \cdot n_0 + m_0$ so R always contains at least one element.
Thus, R is a set of natural numbers and is non-empty. By WOP, R contains a smallest element r_0.
Thus, $r_0 \in \mathbb{R} \implies \exists q_0, m_0 = q_0 n_0 + r_0$.
And r_0 is the smallest element of R, i.e. $\forall r \in R, r_0 \leq r$.
We want to show $r_0 < n_0$.
Suppose $r_0 \geq n_0$.
Then by how r_0 was picked, r_0 is the smallest $r \in \mathbb{R}$. However, if $r_0 \geq n_0$, then there would be a smaller remainder (precisely $r_0 - n_0$) that would contradict r_0 being the smallest element.
In essence:
$m_0 = q_0 n_0 + r_0$
$m_0 = (q_0 + 1)n_0 + (r_0 - n_0)$ if $r_0 \geq n_0$.
Contradicting r_0 being the smallest element of R.
Or more formally, $r_0 \leq r_0 - n_0$ and $r_0 > r_0 - n_0$ are simultaneously true, which is a contradiction.
Thus $r_0 < n_0$.
Exercise:
Plug in numbers.

Claim:
Any statement that can be proved using simple induction or complete induction or well-ordering can be proved using any one of the principles.
How?
Show PSI $\iff$ PCI $\iff$ WOP $\iff$ PSI ...
Show

1. PSI $\implies$ PCI
2. PCI $\implies$ WOP
3. WOP $\implies$ PSI

1\. Assume PSI holds for all predicates. We want to prove PCI.
Let $P : \mathbb{N} \to \{T, F\}$ be a predicate.
Assume $\forall n, (\forall k < nP(k)) \implies P(n)$.
WTS: $\forall n, P(n)$.
omitted due to not paying attn in lecture

2 Algorithm Analysis

Algorithm analysis (AA) is the field of analyzing algorithms. Wow. We can analyze:

- Correctness
- Runtime (time complexity)
- Memory (space complexity)
- Communication complexity (e.g. distributed services)
- and more...

Runtime is the main focus for now.
Runtime measures basic operations and is expressed as a function of the input size.
However, this is under the assumption that the function behaves predictably for inputs. If not, we can study worst-case/best-case/average-case runtimes.
Runtime uses asymptotic notation ($\mathbf{O}, \Omega, \Theta$) to prove bounds.

2.1 Algorithm Correctness

Idea: Instead of designing an algorithm and proving its correctness, design an algorithm from a correct proof.
We can do this with:

- Preconditions
  - Intuitively, describes "valid" inputs
  - Formally, a precondition is something we assume to be true before running the code
  - This can be abused with impractically strong preconditions. Thus, we want weak preconditions for minimal constraints.
- Postconditions
  - Intuitively, describes "expected" outputs
  - Formally, a postcondition is something we assume to be true after running the code
  - In practice, we want to maximize the relevant details we know about the postconditions. Thus, we want strong postconditions.

We see that changing the pre/post conditions for any algorithm changes the correctness of said algorithm.
Thus, we can formally define correctness as:
For every input, preconditions hold $\implies$ code terminates and postconditions hold.

Example 2.1 *Consider the coin denomination proof.*

We define a function `Change(n)` with preconditions: $n \in \mathbb{N}, n \geq 12$ and postconditions: `Change(n)` $=$ (a, b) where a is the number of 3-cent coins and b is the number of 5-cent coins.

```
Change(n):
    if n == 12: return (4, 0)
    if n == 13: return (2, 1)
    if n == 14: return (0, 2)
    if n >= 15:
        (a, b) = Change(n - 3)
        return (a + 1, b)
```

We can now state correctness. For each $n \in \mathbb{N}, n \geq 12 \implies$ `Change(n)` terminates and returns `(a, b)` such that $n = 3a + 7b$.

Notes 2.1.1

1. For recursive algorithms, prove by induction on input size n, usually with complete induction.
2. Must justify that the recursive call is made on a smaller input that satisfies precondtions.