CS571

- Notes 12
- Partial and total correctness

A partial correctness proof that is “incorrect”

- Some proofs have pre-conditions that are too weak
- They are dependent on the “if the program terminates” condition
- E.g. the division by repeated subtraction algorithm

```plaintext
q := 0;
r := x;
while r greater y minus 1 do
  q := q plus 1;
r := r minus y
end
```

The invariant is:

\( \{ x=q\cdot y+r \mid r \geq 0 \} \)

Partial correctness of the division algorithm

- With this invariant we can prove the specification:

\( \{ x \geq 0, x=q\cdot y+r \land y>r \geq 0 \} \)
- This is clearly correct, but what about a constraint on y?
- If y is -ve, the loop diverges (r grows, rather than reducing)
- Really we need the added constraint: \( y \geq 1 \) to strengthen the pre-condition
- The post-condition remains the same, and the proof is essentially the same

Total correctness

- We can actually prove that the loop terminates by finding an expression just like \( r \geq 0 \) in the previous example
- Floyd’s method is in three parts:
  1. Prove partial correctness
  2. Find an expression that can be mapped onto the natural numbers
  3. Show that the value of the expression starts out +ve and reduces to zero every time through the loop body
Well-foundedness

- This works because the natural numbers are a well-founded set
- Properties of a well-founded set:
  - Members can be compared with a relation > (or <)
    - > is transitive: a > b ∧ b > c ⇒ a > c
    - > is asymmetric: a > b ⇒ ¬(b > a)
    - > is irreflexive: ¬(a > a)
  - There is a least element: ∀x∃y.x > y (x≠y)

Examples of well-founded sets

- Natural numbers: 0 < 1 < 2 < 3 < 4..., where < is <
- Substrings of a finite string: “” < “c” < “bc” < “abc”..., where < is substring
- Proper subsets of a finite set: {} < {a} < {a, b} < {a, b, c}..., where < is ⊂

Total correctness of exponentiation

```
e := 1;
t := y;
while t greater 0 do
  e := e mult x;
t := t minus 1
end
```

- Partial correctness first
- Specification is
  \[ y ≥ 0, e = x ** y \]
- Invariant is:
  \[ e = x ** (y-t) ∧ y ≥ t ≥ 0 \]

Floyd expression

- Trivial in this case: just t
- We can prove it by proving \( t ≥ 0 \) based on the initial pre-condition and the loop entry condition
- i.e. \( p_0, l ∧ b ∧ F ≥ 0 \) (\( \vdash \) is the provability relation)
- The reasoning is that if we can prove the well-foundedness of the Floyd expression from the initial pre-condition and it is true inside the loop, then it is the right expression
- In this case: \( y ≥ 0, l ∧ t > 0 ∧ t ≥ 0 \)
- This is trivially true since the invariant contains \( t ≥ 0 \)
Proving loop termination

- In general, we need to examine all possible paths through the loop body, and show that the Floyd expression reduces in value along every path. Again based on the initial pre-condition and the loop entry condition.
- i.e. \( p_0, I \land b \vdash \text{F}_{\text{path start}} > \text{F}_{\text{path end}}, \) for all paths
- In this case there is only one path, so we need to show: \( y \geq 0, I \land t > 0 \vdash t > [t-1]t \)
- This is trivially true because \( t > t-1 \) for any \( t \)
- Thus the loop will terminate with the value of the Floyd expression = 0, i.e \( t=0 \)