Black Book of Bogus Proofs

1 Introduction

This document contains some examples of bad proofs. You should emulate these proofs if you want to convince someone of something that is false. Note that you cannot convince the CS3110 staff of things that are false.

Note: do not read the "Compendium of Correct Proofs".

2 Unproofs of correctness

2.1 Every number is even

We aim to prove the proposition $P(n)$ that n is even.

Base Case: $P(0)$ is trivially true.

Inductive Step: Using strong induction, assume $P(k)$ is true for all $k < n$. Consider $n + 1$. Since both $P(1)$ and $P(n)$ hold and the sum of two even numbers is an even number, $P(n + 1)$ holds. Therefore, all positive integers are even.

2.2 Length is positive

```
(** returns the length of l *)
let rec length l = match l with
  | []     -> -1
  | h::tl -> 1 + length tl
```

Claim: For all lists `l`, `length l >= 0`.

Proof: `length l` computes the length of `l`. Since every list has a non-negative length, `length l >= 0`.

2.3 is_prime is correct.

```
(** returns true if and only if n is prime *)
(*  fails if n <= 1 *)
let rec is_prime n =
  if n <= 1 then failwith "yolo"
  else List.mem n [2;3] || is_prime (n-2)
```

Claim: `is_prime` satisfies its spec; that is, `is_prime n` returns `true` if and only if `n` is prime.

Proof 1: We must prove that (i) if `is_prime n` is `true` then `n` is prime, and also that (ii) if `is_prime n` is `false` then `n` is not prime.

(i) Choose an arbitrary `n`, for example, `n = 17`. By the substitution model, `is_prime n = is_prime 17 --> true`. Moreover, 17 is clearly prime, so case (i) is complete.

(ii) Choose an arbitrary `n`, such as 4. By the substitution model, `is_prime 4 --> false`. Moreover, $4 = 2 \cdot 2$, so 4 is not prime. Thus case (ii) is complete.

Proof 2: Suppose `n` is prime. Then either `n = 2` (in which case `is_prime n` is clearly `true`), or `n = 2*k+1` for some `k >= 2`.

We will prove by induction on `k` that `is_prime (2*k + 1) -->* true`.

Base case: if `k = 1` then `n = 3`. Clearly `is_prime 3` returns true.

Inductive step: Assume that `is_prime (2*k+1) -->* true`. By the substitution model,

$$
\begin{aligned}
\texttt{is\_prime (2*(k+1)+1)} &\to^* \texttt{is\_prime (2*k+3)} \\
&\to^* \texttt{false || is\_prime (2*k + 3 - 2)} && \text{because } k \geq 2 \\
&\to^* \texttt{is\_prime 2*k+1} \\
&\to^* \texttt{true} && \text{by the inductive hypothesis}
\end{aligned}
$$

This completes the proof.

2.4 Every even number is a power of 2

Claim: If n is an even number greater than or equal to 2, then there exists i such that $n = 2^i$.

Proof. Let $P(n)$ denote the statement "if n is an even number greater than or equal to 2, then there exists some i such that $n = 2^i$. We will show that $P(n)$ holds for all n by mathematical induction.

- Base case: $P(0)$ holds since 0 is not greater than or equal to 2.
- Inductive step: We must show that $P(n)$ implies $P(n+1)$. We analyze several subcases. If n is 1 then $P(1)$ holds since $1 = 2^0$. If n is 2 then $P(2)$ holds since $2 = 2^1$. Otherwise, if n is odd, then the assumption that n is even is false and $P(n)$ vacuously holds. Otherwise, n is even, so there exists an m such that $n = 2m$. By induction hypothesis, there exists j such that $m = 2j$. Hence, we have,

$$n = 2m = 2 \cdot 2^j = 2^{j+1}$$

  which finishes the sub-case and the inductive proof.

3 Unproofs of performance

```
let rec merge (left: 'a list) (right: 'a list): 'a list =
  match (left, right) with
    ([],_) -> right
  | (_,[]) -> left
  | (x::rest_left, y::rest_right) ->
      if x > y then y::(merge left rest_right)
               else x::(merge rest_left right)
```

Let $T(n)$ be the worst-case running time of `merge left right` where n is `length left + length right`.

Claim: $T(n)$ is $O(1)$. Proof: By induction on n.

Base case: If $n = 0$, then `left = right = []`. In this case, `merge left right -->* []` in a constant number of steps (say c steps). $T(n) = c$ is constant, so it is $O(1)$.

Inductive step: Suppose $T(n)$ is $O(1)$. We will show that $T(n+1)$ is also $O(1)$. There are three cases: either `left = []`, `right = []`, or neither `left` nor `right` is `[]`.

In the first two cases, `merge left right` steps to either `left` or `right` in a constant number of steps.

In the latter case, the third match branch is taken, and the expressions evaluates (in a constant number of steps, say c_1) to either `y::(merge left rest_right)` or to `x::(merge rest_left right)`. By the inductive hypothesis, either call to `merge` will run in $O(1)$ time. The `::` operation also operates in constant time (call this constant c_2). Putting this all together, we have that the whole call to `merge` runs in

$$T(n) = c_1 + O(1) + c_2 = O(1)$$

time.

Therefore, by induction, for all n, $T(n)$ is $O(1)$.