

# Logic and Computation Notes

William Traub

## Contents

<b>1</b>	<b>1/7, Defining Functions</b>	<b>1</b>
1.1	Course Plan . . . . .	1
<b>2</b>	<b>1/8 Functional Programming</b>	<b>2</b>
2.1	OCaml notes . . . . .	2

## 1 1/7, Defining Functions

mk-requests:(ListID)

$\forall S, \forall r, \forall a, \exists R, \text{mk-requests}(S, r, a) = R$

$\forall S, \forall r, \forall a, \forall R, \text{mk-requests}(S, r, a) = R \implies \Sigma_{(\text{fr}, \text{to}, \text{amt})} \text{amt} = a$

$\forall S, \forall r, \forall a, a < 0 \implies \nexists R, \text{mk-requests}(S, r, a) = R$

Advantages of proofs:

- Computable - we can turn these statements into tests and proofs
- Independent - decoupled from the code itself. We don't need to worry about runtime in order to test.

Coding a property?

Because we are universally quantifying, we would generally need to enumerate over the entire set of each  $\forall$

Write a testing harness to substitute for the enumeration. Use randomness to generate test data.

### 1.1 Course Plan

Modules:

1. Programming and PBT
2. Engineering with Abstractions
3. Analyzing Programming Languages

## 2 1/8 Functional Programming

What is functional programming?

When we write software and it gets complicated, it helps to break it down into smaller pieces. With functional programming, we get 'glue' where you are composing functions with each other.

Goal: get us to write a fizzbuzz function that takes in an int and gives a fizzbuzz string output

---

```
function fizzbuzz:
  input n number
  output string

  if (n % 3) = (n % 5) = 0 return "fizzbuzz"
  else if n % 3 = 0 return "fizz"
  else if n % 5 = 0 return "buzz"
  else return n as a string
```

---

### 2.1 OCaml notes

- in the REPL (utop) terminate expressions with ;;
- standard arithmetic, carrot is string concat
- must use +. to add floats
- use float of int to cast float to int
- double quotes for string
- not binds tighter than other
- if false then 2 else 3 (ternary must have same type)
- comparison operators, single =
- let x = 4 in x / 2
- overwriting variables instead of mutating them
- ---

```
let <var> = <expr> in <body> ;;
< store, let <var> = <expr>> -> <store[<var> ;;
```

---