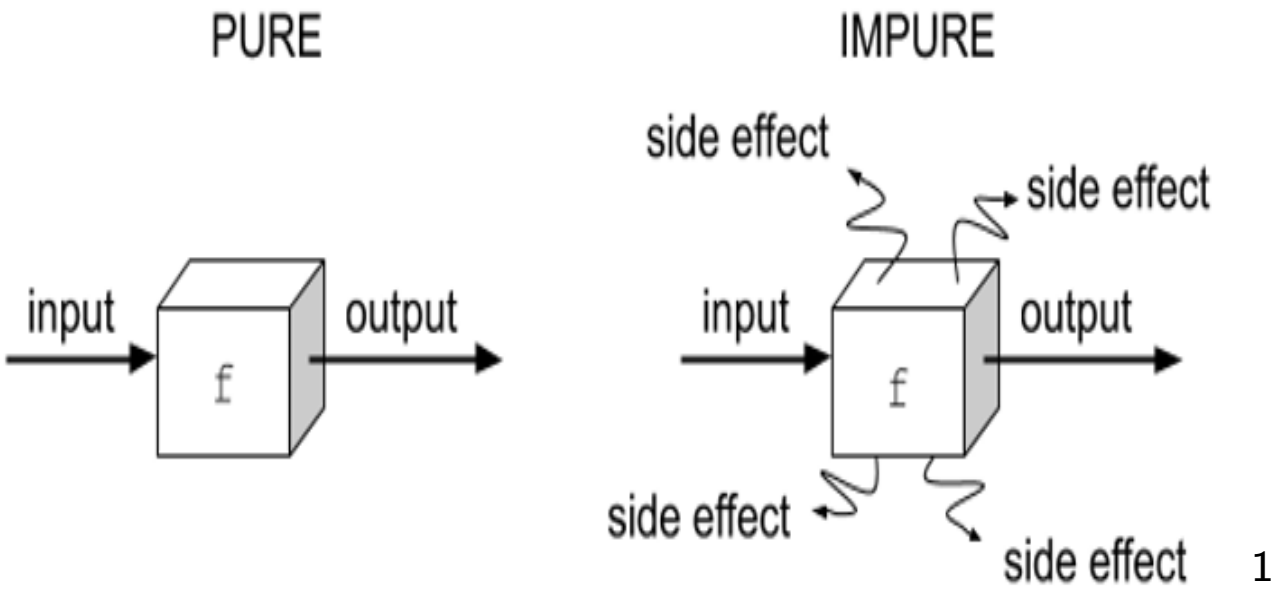
Program Correctness

What is a correct program?

- Correct expressions, statements.
- Calculate what is supposed to do.
- No bugs.
- Properties.
- Specification.
- Verification.

Pure vs Sides Effects



[1]

[1] https://nikhilsukhani.medium.com/understanding-pure-and-impure-functions-ab82965a2c6d

Pure vs Sides Effects

Pure Languages (Racket?)

Checkout Racket reference
`https://docs.racket-Lang.org/reference/` for function definitions.

- Allowed input/out, no other behaviors, predictable.
- Functional languages: Haskell, OCaml, Schem, Scamper etc.

Languages with side effects

- I/O, change something along the way.
- Imperative languages: Python, Java, C#, C, C++, etc.

  Minimize the number of side-effects in your code.

Specify Program Correctness

```
(define list-append
(lambda (l1 l2)
  (if (null? l1)
      l2
      (cons (car l1) (list-append (cdr l1) l2)))))
```

```
(define list-op
(lambda (l1 l2)
  (if (null? l1)
      l2
      (cons 1 (list-append (cdr l1) l2)))))
```

- What kind of properties hold of a program? $\rightarrow$ Specification.
- How do you check? $\rightarrow$ Model of the program.

A Substitutive Model of Computation

Grammar Syntax

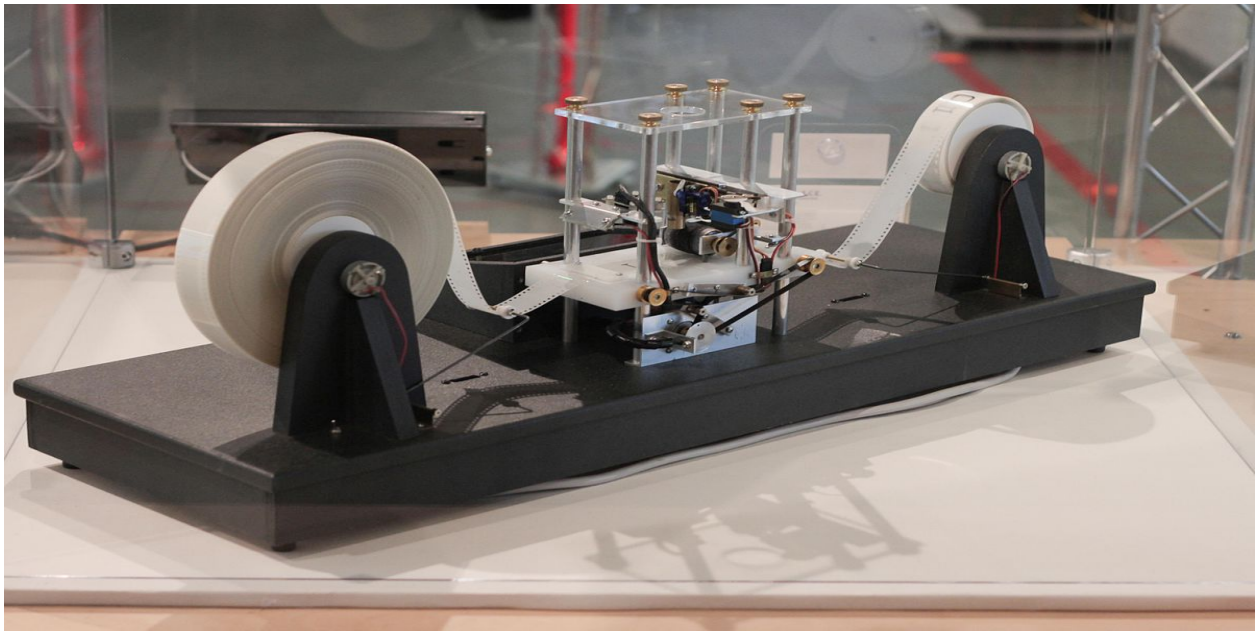
```
Human ::= Boy
        | Girl
        | Baby
        | Adult
        | Happy (Human)      (Emotions)
        | Playing (Human)    (Activities)
Playing ::= Ski | Hike | Workout | Climb
(What kind of sports Human can do?)
```

Operational Semantics

Tells the meaning of operations, symbols in your model syntax.

- Happy Boy $\rightarrow$ "?" maybe :).
- $1 + 1 \rightarrow 1$ (boolean algebra).

Simulation in discrete manner



AND	10001010000	11	1104		R - format
ADD	10001011000	11	1112		R - format
ADDI	1001000100	10	1160	1161	I - format
ANDI	1001001000	10	1168	1169	I - format
BL	100101	6	1184	1215	B - format
ORR	10101010000	11	1360		R - format
ADDS	10101011000	11	1368		R - format
ADDIS	1011000100	10	1416	1417	I - format

A Substitutive Model of Computation

Grammar Syntax

```
e ::= x                                  (variables)
    | <number>                           (numbers)
    | #t                                 (true)
    | #f                                 (false)
    | (lambda (x1 ... xk) e)             (lambdas)
    | (e e1 ... ek)                      (func. app.)
    | (if e1 e2 e3)                      (conditionals)
    | (let* ([x1 e1] ... [xk ek]) e)     (let-bindings)
```

Semantics

```
    (inc (+ 3 (* 5 2)))
--> (inc (+ 3 10))
--> (inc 13)
```

Q & A