1-1) Essentials of Compilation - Jeremy Siek

compiler : A ⇒ B

A is the type of programs of lang X

B is B lang Y

gcc : C ⇒ X86

clang : C ⇒ LLVM

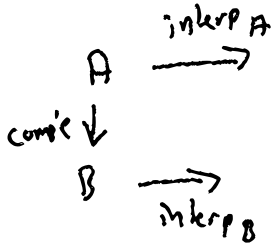
llvm : LLVM ⇒ X86
ARM

When is a compiler correct?

compile : $A \Rightarrow B$

$interp_A$: $A \Rightarrow answer$

$interp_B$: $B \Rightarrow answer$



$\forall p \in A.$

$interp_A(p)$

$=$

$interp_B(compile(p))$

1-3)

When is a compiler "good"?

there is a big "gap" between A & B

two different compilers

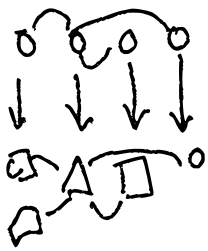
$C_1 : A \to B$ $\quad\quad$ $m : B \to N$

$C_2 : A \to B$

$C_1 > C_2$ iff $m(C_1(p)) > m(C_2(p))$

1-4

// compiler : A ⇒ B



A'

R_0 : p ::= (program info e)

e ::= number
| (= e)
| (+ e e)
| (read)

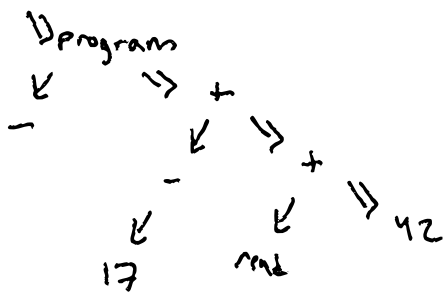
1-5)

S-expressions

"return -17 + read() + 42;"

=>

(program - (+ (- 17) (+ (read) 42)))




```
new Program (
  true,
  new Add (
    new Negate (
      new Num (17)),
    new Add (
      new Read(),
      new Num (42))))
```

["program", true, ["+", ["-", 17], ["+", ["rd"], 42]]

1-b) real answers

interp : $R_0 \rightarrow int$

interp (Num n) = n

interp (Neg e) = -1 * interp (e)

interp (Add l r) = interp l + interp r

interp (Read) = ① ask the user, scanf

② go grab one of 'em

1-7)

$2^n : nt \to R_0$

$interp(2^n(m)) = 2^m$

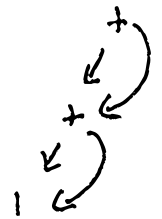
$2^n\ 0 = 1 \quad (Num\ 1)$

$2^n(1+m) = (+\ (2^n\ m)\ (2^n\ m)$

$= new\ Add(x, x)$

$where\ x = 2^n\ m$

$2^n\ 2 =$



1-8)

randp : int => R0

randp 0 = flip a coin heads => (read)
tails => num ∈ (0, 1024)

randp (1+n) : flip a coin heads => (- (randp n)
tails => (+ (randp n)
(randp n))

randp 2 = (- (randp 1))
= (- (+ (randp 0) (randp 0)))
= (- (+ (rd) (2))

1-9)

optimizer : $R_0 \Rightarrow R_0$

opt : $R_0 \to R_0$

```
opt n = n
opt read = read
opt (- e) = not-opt (opt e)
not-opt = cases
    Num n -> Num (-1 * n)
    Neg e' -> e'
    Add (Num n) e' -> Add not-opt (Num n)
                          not-opt e'
    e' -> Neg e'
```

1-10)

```
opt (Add (l, r)) = case (opt l, opt r)
  (Num ln, Num rn) => Num (ln + rn)
  (Num ln, Add (Num rn) re) =>
        Add (Num (ln + rn)) re
  (Add (Num ln) le, (Num rn)) =>
        Add (Num (ln + rn)) le
  (Add (Num ln) le, Add (Num rn) re) =>
      Add (Num (ln + rn)) (Add le re)
  (le, Num rn) => Add (Num rn) le
  (le, re) => Add le re
```

1-11

opt (- (+ 10 (+ (read) 12)))

↓

(+ 12 (read))

↙

(+ 22 (read))

↙

(+ -22 (- (read)))

```
for i in 0 to 1024; do
    p = randp 6
    p' = opt p
    n = count-rads p
    ans = generate-Randoms n
    check (interp^ans p) = (interp^ans p')
```