eval :: String → Answer

eval-py("{'x': [1]}") is [strng-dict: "x", [list: 1]]

eval-java("""class C {...}
new C()""") is class-instance...

String → Answer

parse :: String → Prog

interp :: Prog → Answer

5 + 4 * 6 + 3

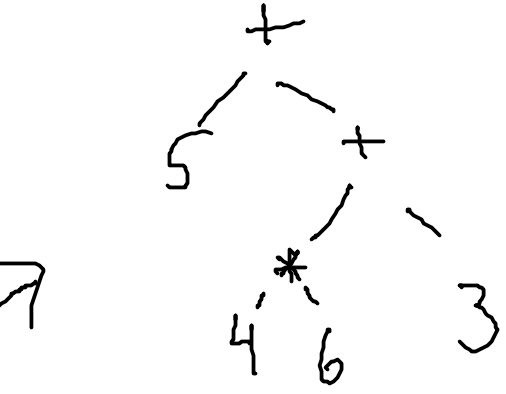
5 + (4 * 6) + 3

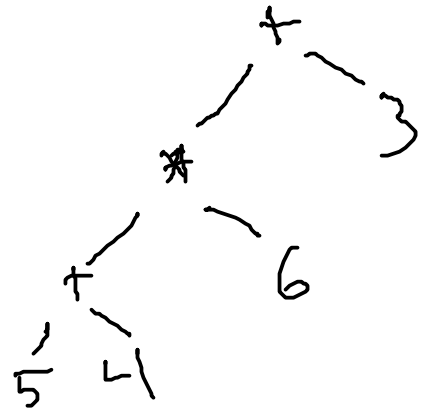
((5 + 4) * 6) + 3

5 4 6 * + 3 +

(+ 5 (+ (* 4 6) 3))

Concrete Syntax





5+4

x-1 — identifier

— binop

HCI

Unambiguous

Easy to parse

Ugly

Verbose

s-exp := number 5.5 ->
| quoted-str "hello"
| symbol x + - / hello
| (s-exp ...)

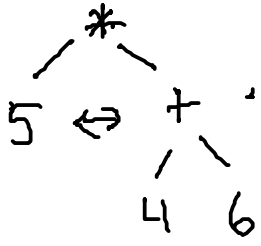
(+ 5 (* 4 6))

Context-free grammar

BNF

(+ 5)

a-exp := number
| (* a-exp a-exp)
| (+ a-exp a-exp)



```
interp :: Prog -> Answers
data AST:
  | multiply(lhs :: AST, rhs :: AST)
  | plus(lhs :: AST, rhs :: AST)
  | num(n :: Number)
end
```

Abstract Syntax Tree

Number ≡ Answers

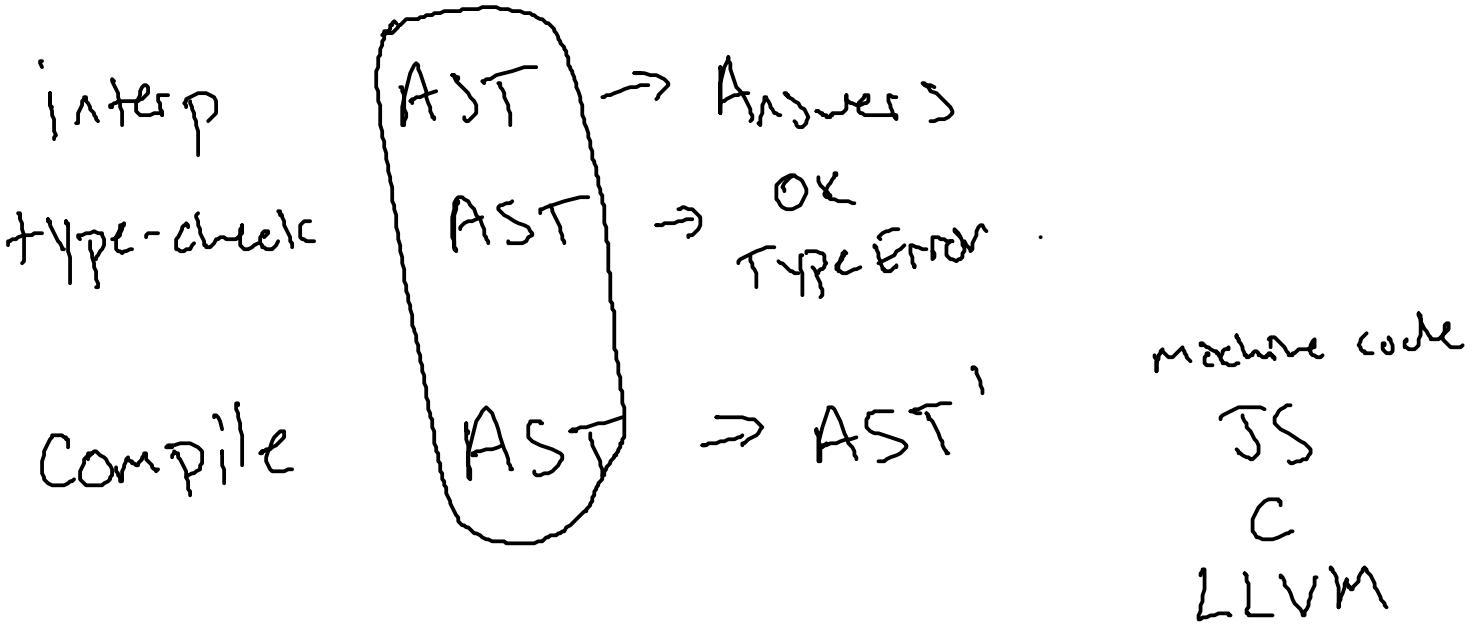
5

multiply(num(5), num(6))

multiply(num(5), plus(4, 6))

multiply(plus(4,6), num(5))

```
fun interp(p :: Prog) -> Answer:
  cases(Prog) p:
    | multiply(l, r) =>
      interp(l) * interp(r)
    | plus(l, r) =>
      interp(l) + interp(r)
    | num(n) => n
  end
end
```


interp
type-check
Compile
AST
AST
AST
→ Answers
→ OK
TypeError
→ AST'
machine code
JS
C
LLVM