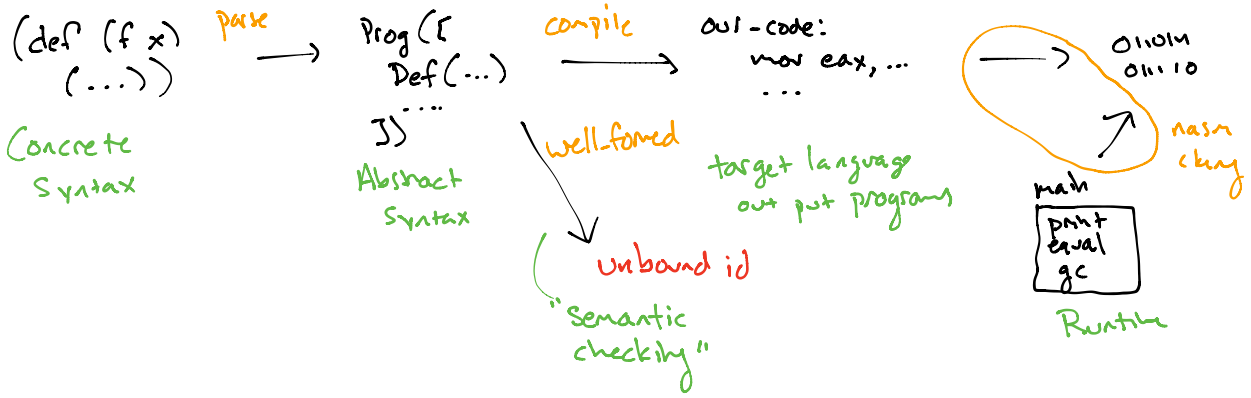


How can we make compile "better"? Generate "better" assembly

AST ⟶ ASM

- can we use registers (for args)?
- assembly instr speed?
- skip unneccessary error fncs
- do some work at compile time
- do work in parallel

goals

- make gen prog faster
- use less memory
- easier to debug
- faster compile time (gen "better" asm)
- smaller binary
- less power

```
let rec compile e ... =
    ["mov eax, 11"]
```

Optimization - a compiler that produces programs with the same answers and effects when run (while making things "better")

same answers: what's in EAX, if there was an error

effects: print the same output

An optimization is an improvement to the compiler that generates "better" target programs (assembly code) with the same *answers* and *effects*.

Which of the following isn't an optimization, by the above definition?

A: In expressions like `(+ x 1)`, omit the tag check for 1 and don't store it on the stack

rhs of is constant

B: In expressions like `(if true 5 6)`, where the conditional position is true, compile only to the instructions in the then/true branch, and omit the else/false branch

C: In expressions like `(+ 10 5)` with constants in the operator, compile to move the result (e.g. 15) into EAX directly

D: In expressions like `(if x 5 6)`, treat all non-false values as true in conditional position

Changes the program's behavior

E: In expressions like `(let (x 5) (+ x 10))`, where the variable is a known constant, generate the instructions that would be generated for `(+ 5 10)`

```
(def (f x)
  (+ x 1))
```

Instructions we don't need?

```
f:
  mov ebp, esp
  sub esp, 8
  mov eax, [ebp + 0]   ← look up x
  mov [ebp - 4], eax
  mov eax, 3
  mov [ebp - 8], eax   } mov [ebp-8], 3
  and eax, [ebp - 4]
  and eax, 1
  cmp eax, 1
  jne near error_non_int
  mov eax, [ebp - 8]
  and eax, 0xfffffffe
  add eax, [ebp - 4]
  jo near overflow_check
  add esp, 12
  ret
```

```
f:
  mov ebp, esp
  mov eax, [ebp + 0]
    tag check eax
  mov eax, [ebp + 0]
  add eax, 2
  jo overflow_check
  ret
```