Compiler Optimizations

Goal: Generate better code!

Requirement: Correctness - “obeying the semantics”

Contract: To produce semantically-equivalent code

Flexibility: Many different x86 programs
Compiler Optimization

= Static Analysis
  \[ \text{without running the program} \]

= Liveness Analysis
  \[ \text{learn something about the program} \]

- Derive facts about the input program, all possible executions of that program without running it.

= Program Transformation
  = Use facts to compile to a semantically-equivalent program

\[ P_0 \text{ print } 1+2 \]
\[ P_1 \text{ print } 1+2 \]

- polymorphism / dynamic typing
  \[ \Rightarrow \text{postponing decision to run-time} \]
  \[ \Rightarrow \text{generating dispatch code} \]
Type Specialization

- Observation: If we know some type information at compile-time, we can prune some branches of the dispatch code.

= Static Analysis

- Going to derive types at compile-time (best effort)

+ Transformation

(pruning using type information)

```
print 1 + 2 : int
```

```
int-plus
```

```
call-to-case-f(c)
```

```
print
```

```
unk/dyn proj
```

```
(2 if input() else x) + 3 : int
```

```
end
```
What about assignments?

\[ x = \begin{cases} 1, & \text{if } f() \\ \frac{[25, 27]}{x_0 + x_1} & \text{else} \end{cases} \]

\[ y = x_0 + x_1 \]

\[ z = x_1 + \lceil x_0 \rceil \]

Flow-sensitive analysis

Static Single Assignment

\[ x_0 = 1 \]

\[ y = x_0 + x_0 \]

\[ x_1 = [25, 27] \]

\[ z = x_1 + \lceil x_0 \rceil \]

Flow-sensitive analysis

if \( f() \):

\[ x_0 = 1 \]

\[ y = x_0 + x_0 \]

\[ x_1 = [25, 27] \]

else

\[ y = x_0 + x_0 \]

\[ y_1 = x_0 + x_1 \]

\[ z = \phi(x_0, x_1) \]

print \( z \)

PMFSA

\[ x_0: \text{int}, 1 + 3 \]

\[ y_0: \text{int} \]

\[ x_1: \text{list} \]

\[ y_1: \text{list} \]

\[ x_2: \text{list}, 1 + 3 \]