Lec 15 Dataflow Analysis

liveness analysis — interference
neededness analysis — dead code
reaching definition — constant / copy propagation

$$\frac{use(\ell, x)}{live(\ell, x)} \qquad \frac{live(\ell', u) \quad succ(\ell, \ell') \quad \neg def(\ell', u)}{live(\ell, u)}$$

Memory Liveness

$$\frac{\ell : M[y] \leftarrow x}{use(\ell, x) \quad \cancel{def(\ell, M[y])} \quad use(\ell, y) \quad succ(\ell, \ell+1)}$$

← but y is a temp and we don't allocate reg for M[y]

$$\frac{\ell : x \leftarrow M[y]}{def(\ell, x) \quad use(\ell, y) \quad succ(\ell, \ell+1)}$$

Dead Code with liveness ?

Dead code : operations (in abs asm) that don't influence the result of a function

Result : mem effects, return value, errors, non-termination

Ex. fact again

			live
ℓ_1	p ← 1		p
ℓ_2	p ← p * x	z is not live at ℓ_4, can change ℓ_3 to nop if ℓ_3 doesn't side effect	p, x
ℓ_3	z ← p + 1		p, x
ℓ_4	x ← x - 1		p, x
ℓ_5	if (x > 0) then ℓ_2 else ℓ_6		p, x
ℓ_6	return p		p

Ex'

		live
ℓ_1	p ← 1	p, z
ℓ_2	p ← p * x	p, x, z
ℓ_3	z ← z + 1	p, x, z ← Doesn't work anymore
ℓ_4	x ← x - 1	p, x, z
ℓ_5	if (x > 0) then ℓ_2 else ℓ_6	p, x, z
ℓ_6	return p	p

Neededness Analysis

→ Propagate from important places like return, mem write, & division

$$\frac{\ell : x \leftarrow y \oplus z}{nec(\ell, z) \quad nec(\ell, y)} \text{ (effectful)} \qquad \frac{\ell : \text{return } x}{nec(\ell, x)}$$

$$\frac{\ell : M[y] \leftarrow x}{nec(\ell, z) \quad nec(\ell, y)} \qquad \frac{\ell : x \leftarrow \cancel{M[y]}}{\cancel{nec(\ell, y)}}$$ not needed in CO because of mem safety

$$\frac{\ell : \text{if } (x \,?\, c) \text{ then } \ell' \text{ else } \ell''}{nec(\ell, x)}$$

$$\frac{nec(\ell, x)}{needed(\ell, x)} \qquad \frac{needed(\ell', x) \quad succ(\ell, \ell') \quad \neg def(\ell', x)}{needed(\ell, x)}$$

$$\frac{needed(\ell', x) \quad succ(\ell, \ell') \quad def(\ell, x), use(\ell, y)}{needed(\ell, y)}$$

ℓ: x ← y
ℓ': return x

Neededness analysis complexity : same as liveness
$O(\#\text{vars} \cdot \#\text{lines})$

Ex

		needed
ℓ_1	p ← 1	x
ℓ_2	p ← p * x	p, x
ℓ_3	z ← z + 1	p, x
ℓ_4	x ← x - 1	p, x
ℓ_5	if (x > 0) then ℓ_2 else ℓ_6	p, x
ℓ_6	return p	p

so z is never needed

start →

*Dealing with things after return: make return not have successor

Ex

$\cancel{\ell_1 : i \leftarrow 0}$
$\cancel{\ell_2 : \text{if } (i < 0) \text{ then error else } \ell_3}$ (0 < 0 false)
$\ell_3 : \text{if } (i \geq n) \text{ then error else } \ell_5$ (0 ≥ n)
$\cancel{\ell_4 : t \leftarrow i + s}$
$\ell_5 : u \leftarrow a + t$ (t = s)
$\ell_6 : x \leftarrow M[u]$

① const prop
② const fold
③
④

As part of larger loop:

$\ell_7 : i \leftarrow i + 1$
$\ell_8 : \text{if } (i < n) \text{ then } \ell_2 \text{ else } \ell_9$
$\ell_9 : \text{return } x$

can propagate i = 0 any more
Both ℓ_1 & ℓ_7 reach ℓ_2
→ SSA or implement reaching definitions