5-2

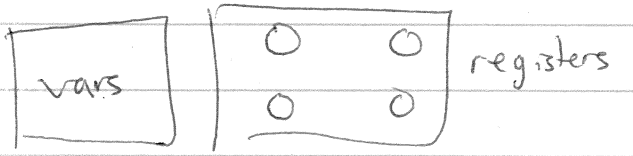
Input : Graph G
Output : Map color from V → REG

W ← V ; i ← 0
while W ≠ ∅ do
  let v be a member of W with smallest
    options(v), largest degree(v), random o.w.
  select c s.t. c ∈ options(v) and c is minimal
  if options(v) empty, then c = stack(i++)
  o.w. color(v) = c
  W ← W - {v}
  → update adjacent neighbors of v for new saturations
    resort of queue

$n \cdot (n + \lg n)$ $O(n^2)$

---

return value is always rax (movq (var x) rax)
mult result is always rcx, rdx (multq (var y) (var z))



callq label → interfers w/ caller-saves registers
rax, rdx, rcx, rsi, rdi, r8, r9, r10, r11

7. push y callee: sp, bp, b, 12-15
(x=10, y=17, z=12)
8. (call label) y ← caller, x, z ← callee
(x=10, y=???, z=12)
9. pop y

5-3 move-biasing

(movq (var x) (var y))

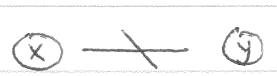
↓

(movq (reg a) (reg b))

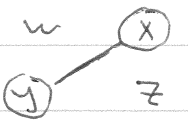
color(x) = a color(y) = b

color(x) = color(y) (assuming they don't interfere)

G:



MOVE-RELATED



7-1 R_2 = e = ... | #t | #f | (cmp e e) | (and e e) | (or e e) | (if e e e)

(c c? t : f) (+ (if #t 1 2) 3)

cmp = eq? | < | ≤ | > | ≥

(+ #t 7) ⇒ 8? error ? not a program

(not 8) ⇒ -6789723 ⇒ raise ex1

```
movq $7, %rax
addq $1, %rax
```

```
(not (if (eq? 0 (read))
         #t
         7))
```

---

after uniquify at a type-checking step

typeof : program (e) x (Var → Type) → type type = bool | int

T(num) = int

T(#t) = bool

T((+ L R)) = int if T(L) = int && T(R) = int

T((≤ L R)) = bool if T(L) = T(R) = int

T((read)) = int

(let ([x #t]) (if x 7 1))

~~T(Γ, x) = Γ(x)~~

T(Γ, (let ([x xe]) b)) = T(Γ[x ↦ T(Γ, xe)], b)

T(Γ, (if c t e)) = T(Γ, t) if T(c) = bool && T(t) = T(e)

7-2

flatten : $R_2 \Rightarrow C_2$

e = (+ e e)
(if e e e)

a = num | var
e = (+ a a) | (- a)
s = (ret a) | (set! x e)
p = s ...

#t/#f → a
and/or/not/cmp → e
if e^R e → if e ∈ C_S ← US s = ... | (if a s s)
↳ x,y,z ∈ C_S ← patch step

(cmp a a)
not ret

flatten (if e_1 e_2 e_3) = (vs, stmts..., ret a)
(vs_1, st_1, r_1) = flatten (e_1) // cond
(vs_2, st_2, r_2) = flatten (e_2) // true
(vs_3, st_3, r_3) = flatten (e_3) // false
< vs = {ans} ∪ $vs_{1,2,3}$
st = st_1 ; (if (eq? r_1 #t) [st_2 ans←r_2] [st_3 ans←r_3])
r = ans
>

optimize with
flattencmp = (vs, st..., (cmp a a))
fc(#t) = (∅, , (eq? #t #t))
fc(#f) = (∅, , (eq? #f #t))
fc((< e_1 e_2)) = (vs_1 ∪ vs_2, st_1 .. st_2 ..,
↳ calls flatten (< a_1 a_2))
(vs_1, st_1, a_1) = f(e_1)
(vs_2, st_2, a_2) = f(e_2)

T(Γ, (if e #t 1))
= type
(int ∪ bool)
⇒ types as sets and predictions

(int ∩ bool) ⇒ types as "what you can do"
⇒ OO

more complicated if
(if (number? x)
(+ 1 x)
(strlen x))
x : (int ∪ str)

T : Γ × e → Ty : int ⇒ Racket
x Props that are true if e is true
x Props that are true if e is false