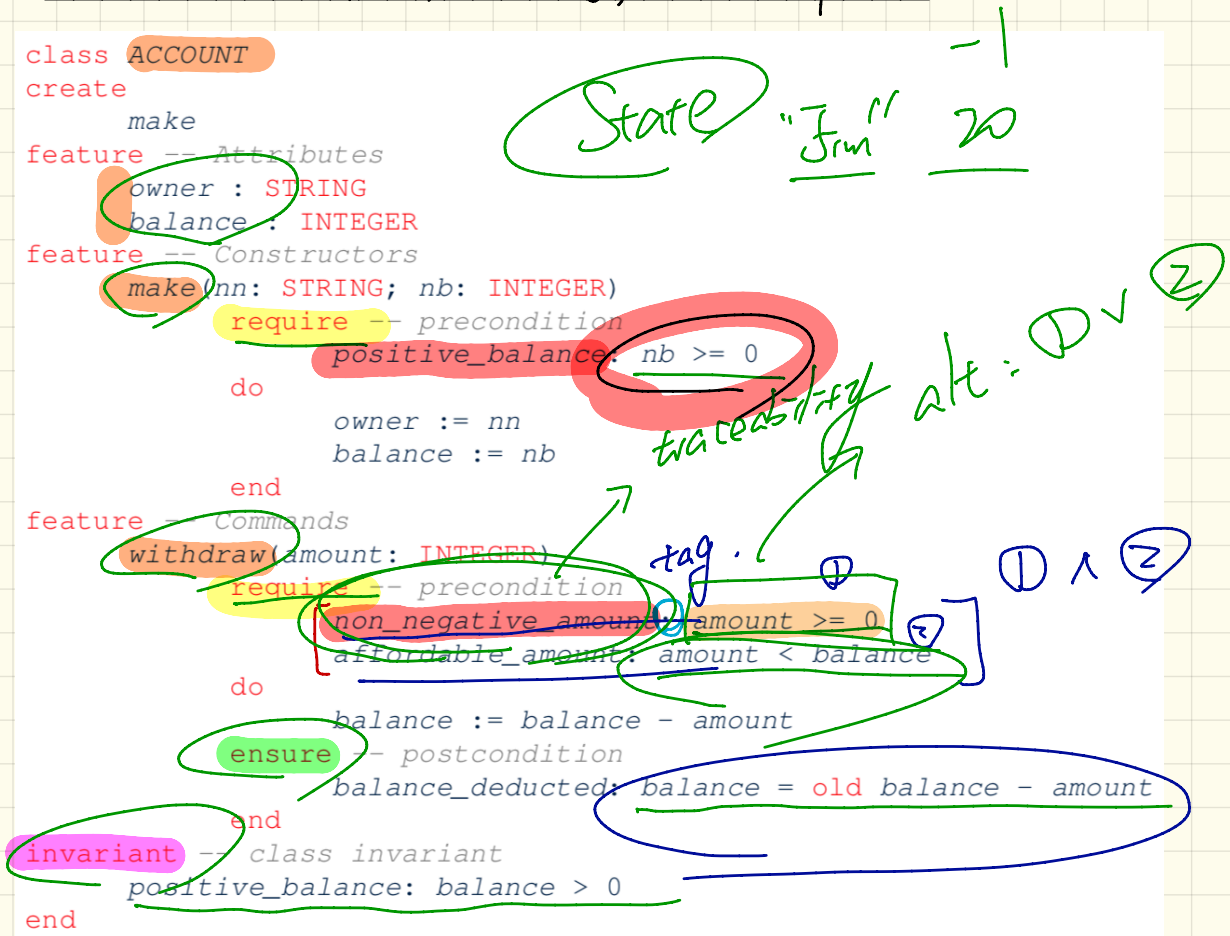
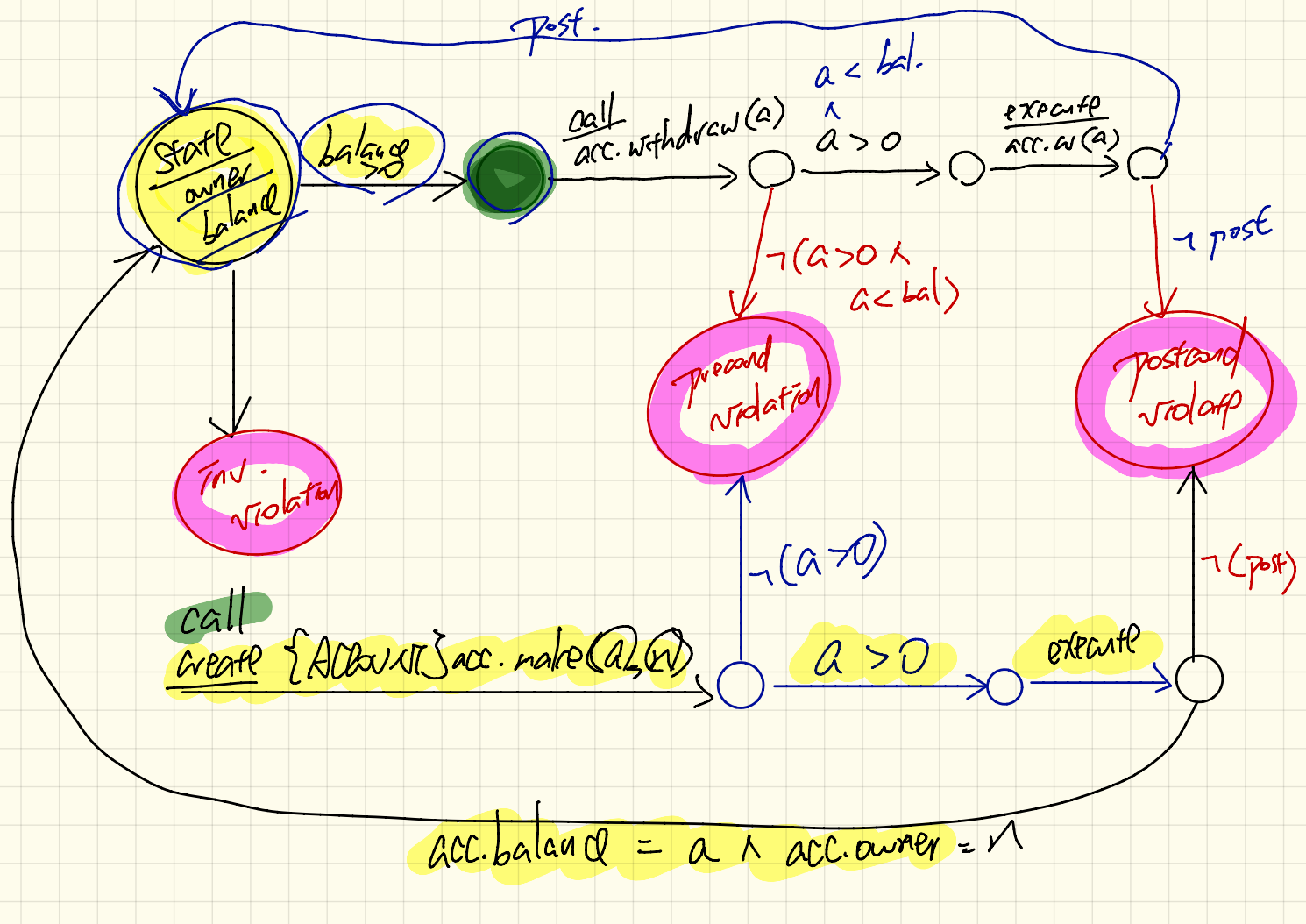
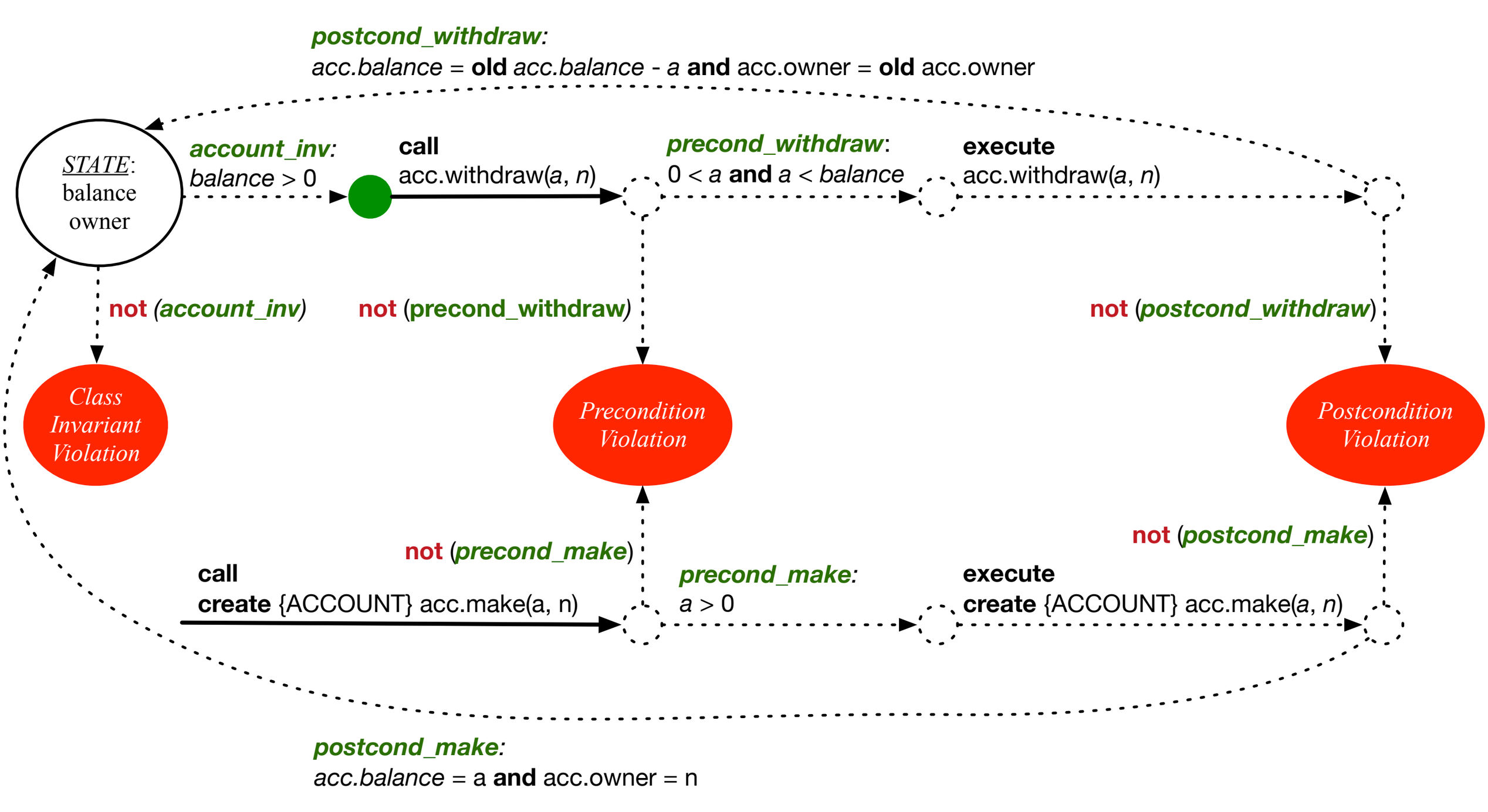
Thursday Sep. 13

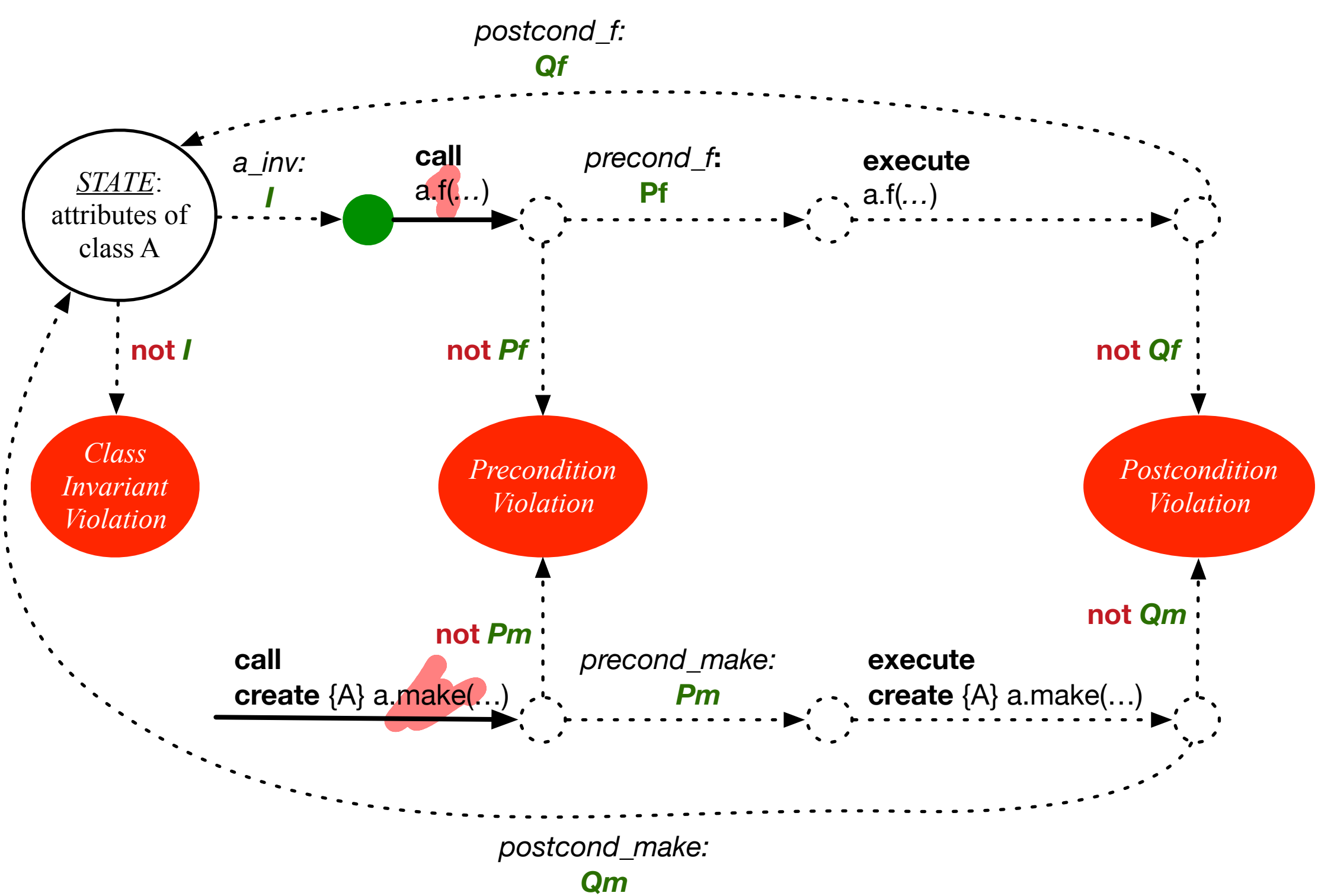
Lecture 3

Account Class: State, Contracts, Creation, Updates




post.
a < bal.
state
owner
balance
balance >0
call
acc.withdraw(a)
a > 0
execute
acc.w(a)
¬(a>0 ∧ a<bal)
¬ post
Inv. violation
Precond violation
Postcond violation
call
create {Account}acc.make(a,v)
¬(a>0)
a > 0
execute
¬(post)
acc.balance = a ∧ acc.owner = v


postcond_withdraw:
acc.balance = old acc.balance - a and acc.owner = old acc.owner
STATE:
balance
owner
account_inv:
balance > 0
call
acc.withdraw(a, n)
precond_withdraw:
0 < a and a < balance
execute
acc.withdraw(a, n)
not (account_inv)
not (precond_withdraw)
not (postcond_withdraw)
Class Invariant Violation
Precondition Violation
Postcondition Violation
not (postcond_make)
not (precond_make)
call
create {ACCOUNT} acc.make(a, n)
precond_make:
a > 0
execute
create {ACCOUNT} acc.make(a, n)
postcond_make:
acc.balance = a and acc.owner = n


postcond_f:
Qf
STATE:
attributes of
class A
a_inv:
I
call
a.f(...)
precond_f:
Pf
execute
a.f(...)
not I
not Pf
not Qf
Class
Invariant
Violation
Precondition
Violation
Postcondition
Violation
not Pm
not Qm
call
create {A} a.make(...)
precond_make:
Pm
execute
create {A} a.make(...)
postcond_make:
Qm

Java

$a = b$

$a == b$

Eiffel

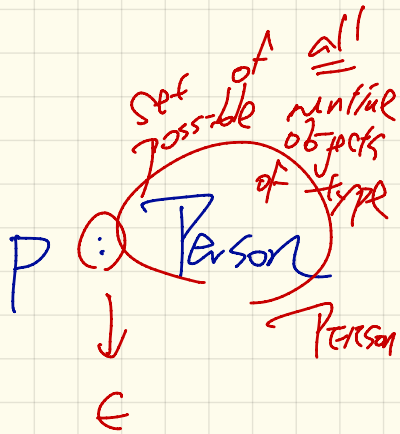
contracts

$a := b$

$a = b$

feature ::
  Name
  Preand
  Local

Person p



Java

a
i
and then

① a[i] >= 0 && 0 <= i && i < a.length

② 0 <= i && i < a.length && a[i] >= 0

∧ Java ×

Eiffel and or implies = ⇔

p	q	p ∧ q
T	T	
T	F	
F	T	
F	F	

command (a: ARRAY[INTEGER]; i: INTEGER)

require

rebase.

v1

not_too_small: i >= 0

not_too_big: i < a.count

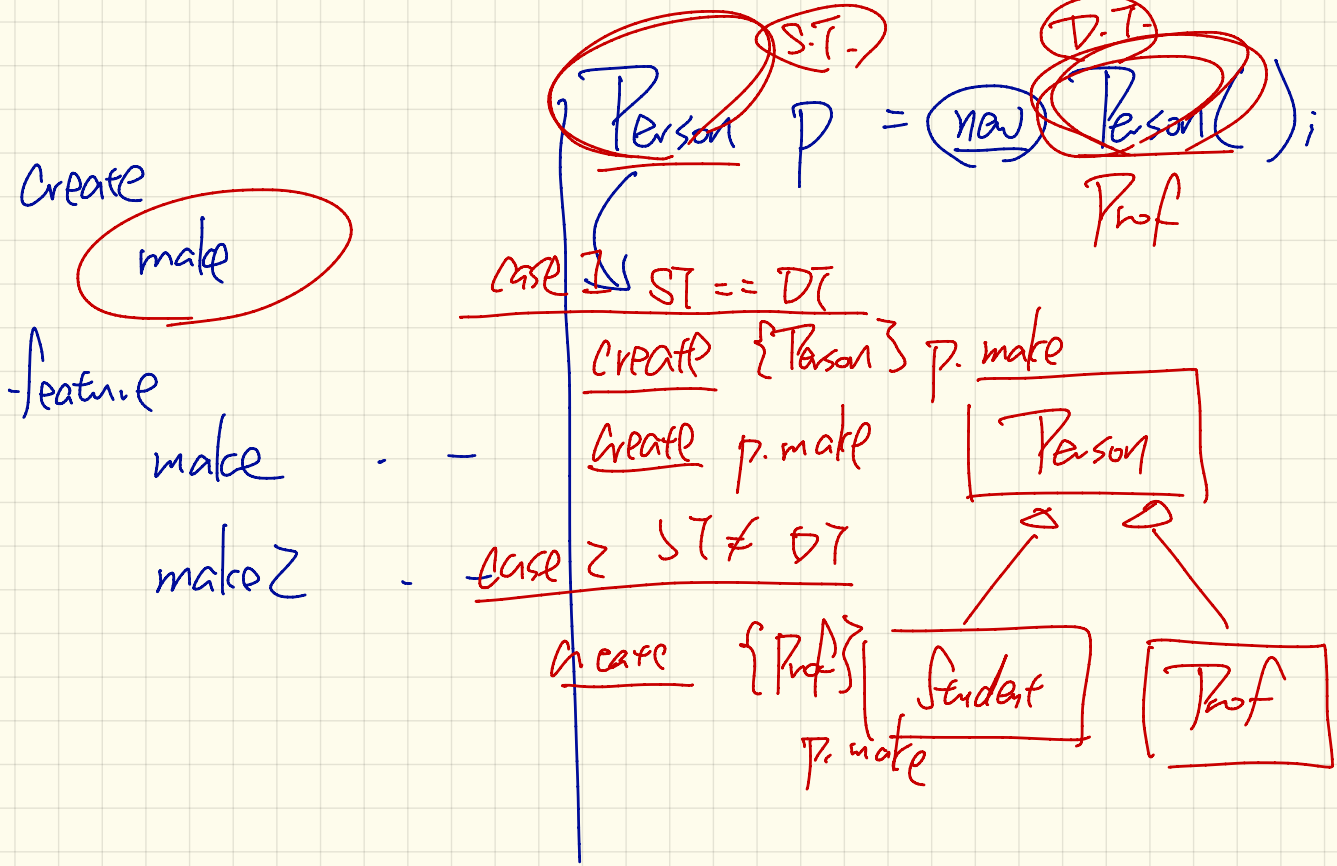
v2 ✗

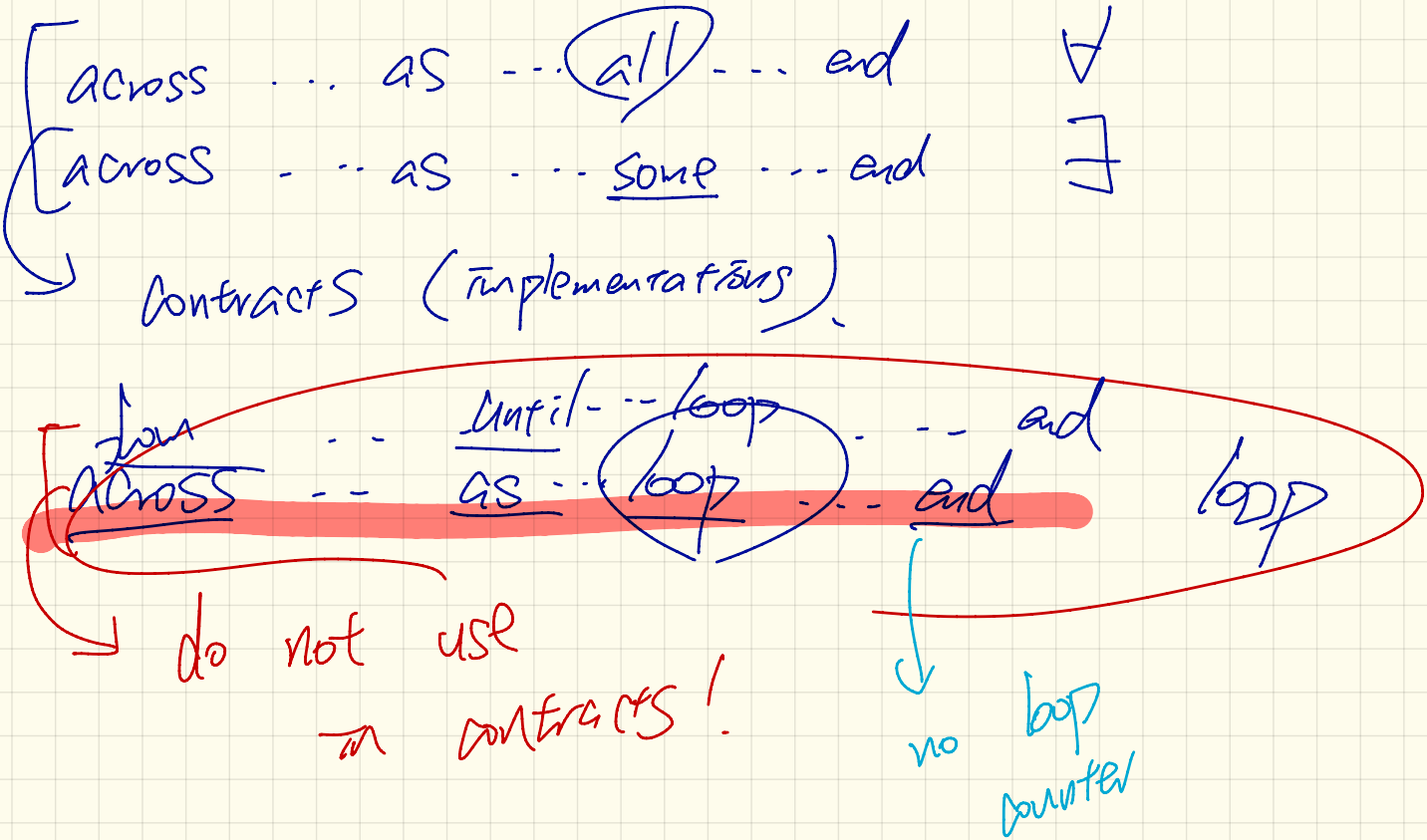
positive: a[i] > 0

pos_element: i >= 0 and then i < a.c and then a[i] > 0

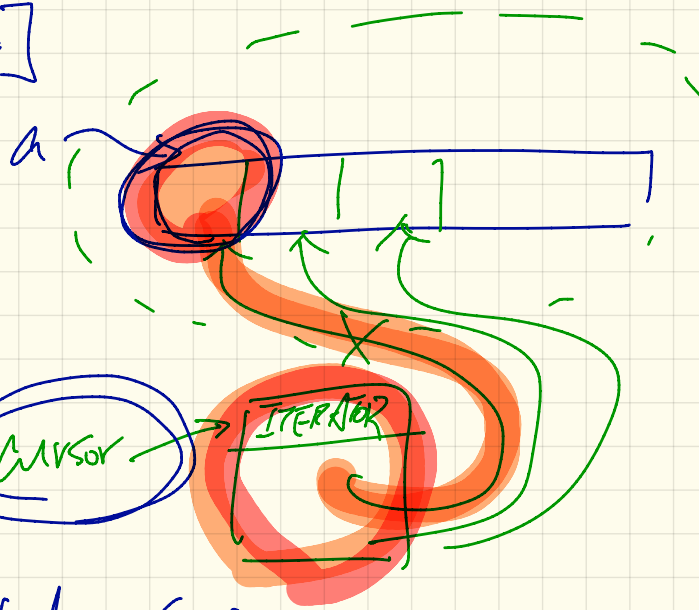
i >= 0 and i < a.count

and a[i] > 0


S.T.
D.T.
Person p = new Person();
Prof
create
make
-feature
make
make2
case 1 ST == DT
create {Person} p.make
create p.make
case 2 ST ≠ DT
create {Prof}
p.make
Person
Student
Prof


across ... as ... all ... end
across ... as ... some ... end
contracts (implementations)
from ... until ... loop ... end
across ... as ... loop ... end
loop
do not use
in contracts!
no loop counter

a: ARRAY [ACCOUNT]



across a as Cursor loop

Cursor.item.withdraw(10)

end

Cursor.withdraw(10)

```
from
    i := a.lower
until
    i > a.upper
loop
    a[i].withdraw(10)
    i := i + 1
end
```

$\forall i \; \forall j$ $\quad \forall i, j \in low..upp \cdot$

$i \neq j \Rightarrow a[i] \neq a[j]$

across

1 1..1 10 as c1

a.lower a.upper

all

across

all 1 1..1 10 as c2

a.lower a.upper

[c1.item /= c2.item implies

a[c1.item] /= a[c2.item]

end

end