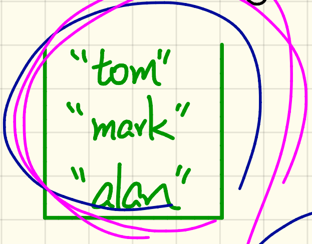
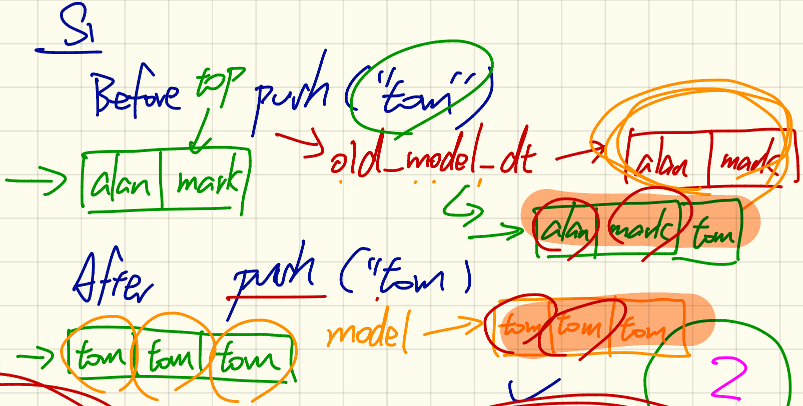
Tuesday Nov. 13

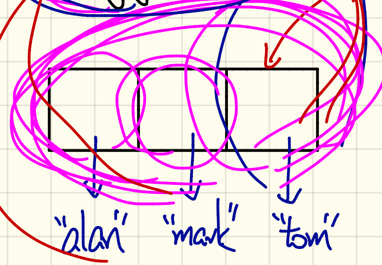
Lecture 18

Implementing a LIFO STACK

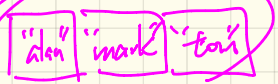




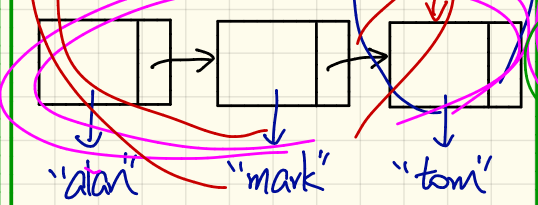
Strategy 1



MODEL

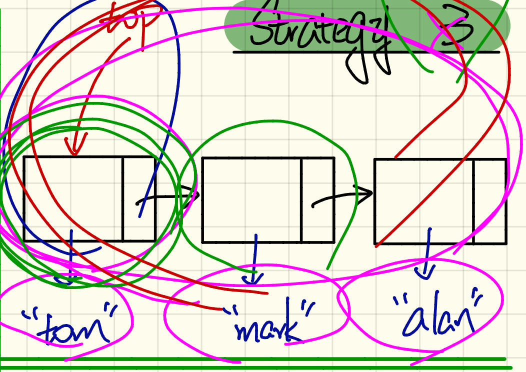


Strategy 2

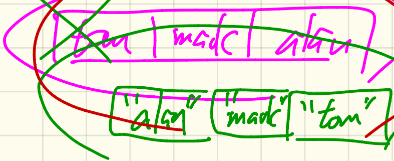


MODEL

Strategy 3



MODEL



Using MATHMODELS Library

(old model).deep_twin). appended(g)

call to the query

Implementing Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
 imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
 model : SEQ[G]
  do create Result.make_empty
    across imp as cursor loop Result.append(cursor.item) end
  end
```

prepend

command

S1

converting from imp into SEQ

Writing Contracts using Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
  push (g: G)
   ensure model ~ (old model.deep_twin).appended(g) end
```

query

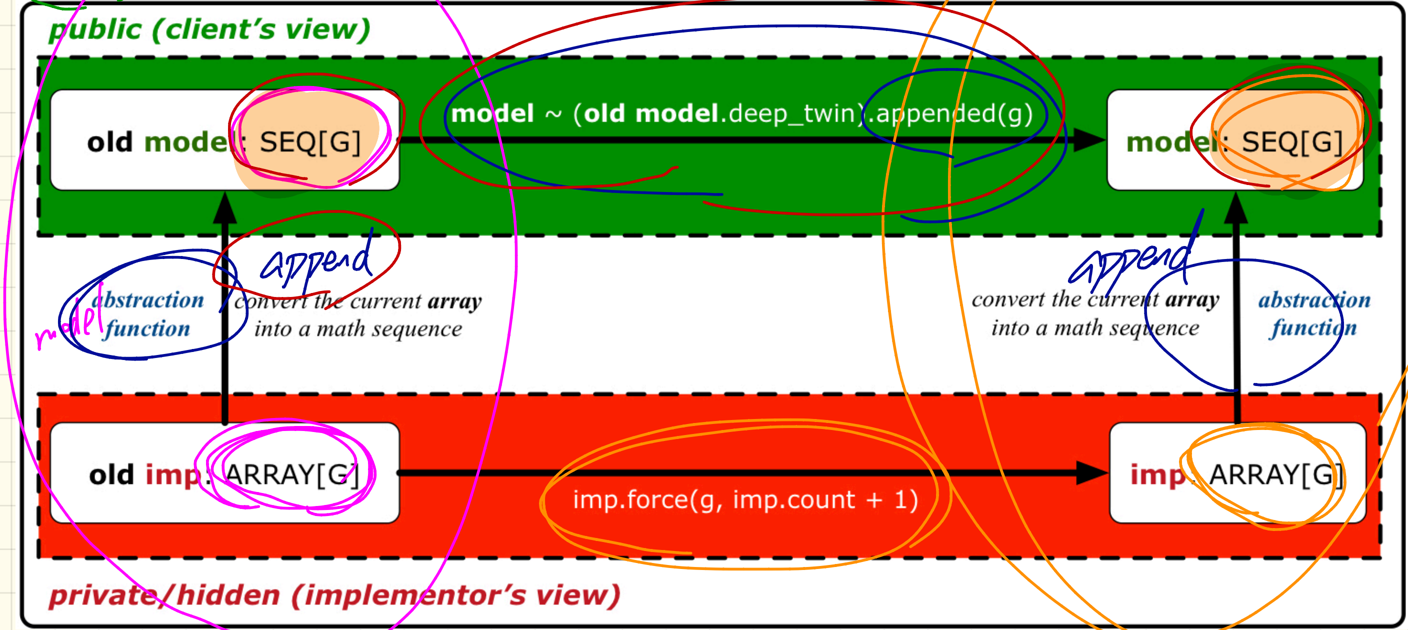
after push is expected

before

immutable query.

Strategy 1 : Mathematical Abstraction

'push(g: G)' feature of LIFO_STACK ADT



public (client's view)

old model: SEQ[G]

model ~ (**old model**.deep_twin).appended(g)

model: SEQ[G]

abstraction function

*convert the current **array** into a math sequence*

append

model

*convert the current **array** into a math sequence*

abstraction function

append

old imp: ARRAY[G]

imp.force(g, imp.count + 1)

imp: ARRAY[G]

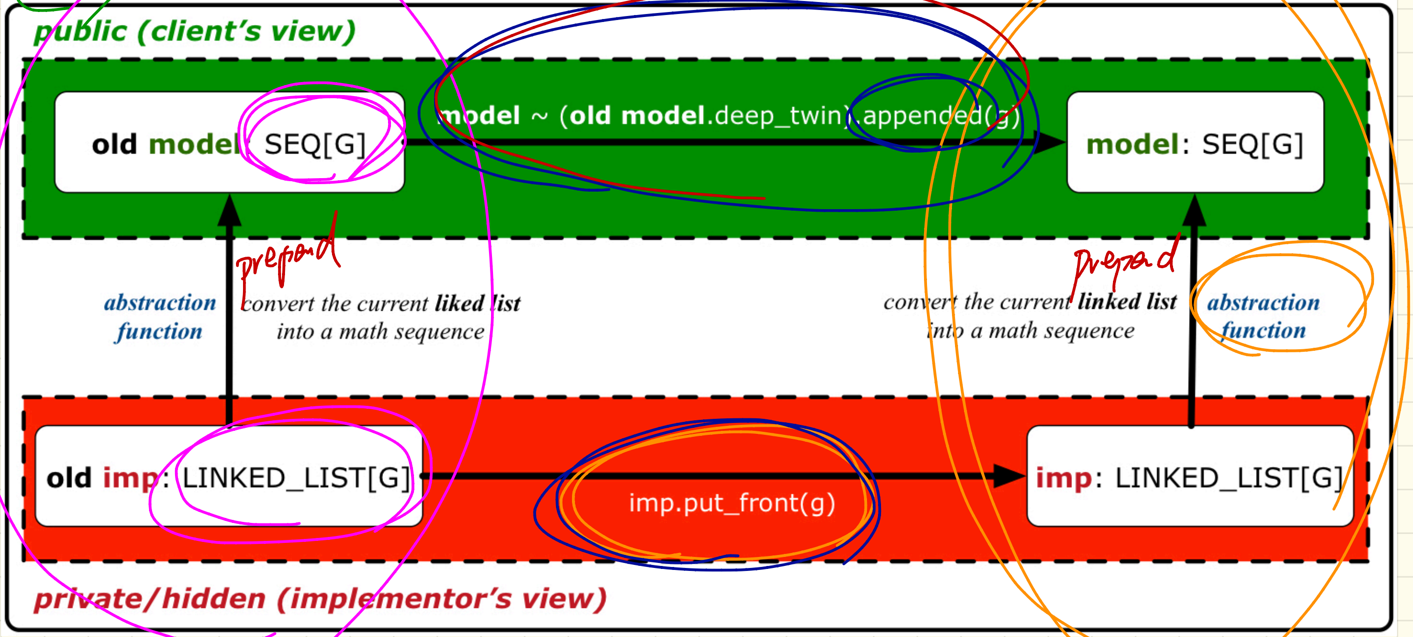
private/hidden (implementor's view)

before

after

Strategy 2: Mathematical Abstraction

'push(g: G)' feature of LIFO_STACK ADT


public (client's view)
old model: SEQ[G]
model ~ (old model.deep_twin).appended(g)
model: SEQ[G]
abstraction function
prepend
convert the current liked list into a math sequence
prepend
convert the current linked list into a math sequence
abstraction function
old imp: LINKED_LIST[G]
imp.put_front(g)
imp: LINKED_LIST[G]
private/hidden (implementor's view)

before

after

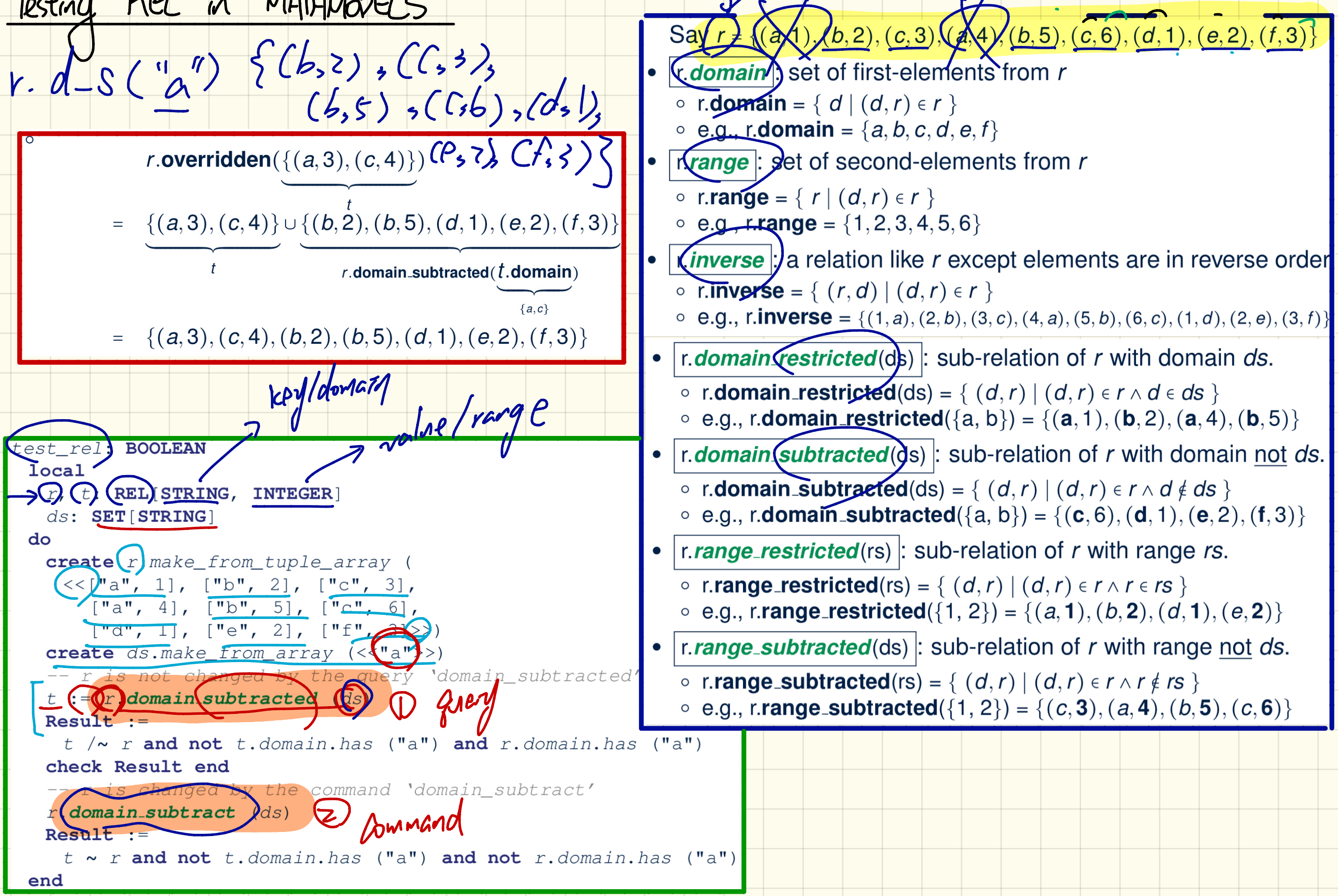
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_from_array (imp)          C1
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make_empty ensure model.count = 0 end
  push (g: G) do imp.force(g, imp.count + 1)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.remove_tail(1)
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 2 (first as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty                     C2
      across imp as cursor loop Result.prepend(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[count - i.item + 1]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.put_front(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.start ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 3 (last as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty                     C3
      across imp as cursor loop Result.append(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.extend(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.finish ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

Testing REL in MATHMODELS

r.d_s("a") {(b,2), (c,3),
(b,5), (c,6), (d,1),



$$r.\textbf{overridden}(\underbrace{\{(a,3),(c,4)\}}_{t})$$ (e,2), (f,3)}

$$= \underbrace{\{(a,3),(c,4)\}}_{t} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{r.\text{domain\_subtracted}(\underbrace{t.\text{domain}}_{\{a,c\}})}$$

$$= \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\}$$

key/domain

value/range

```
test_rel: BOOLEAN
 local
   r, t: REL[STRING, INTEGER]
   ds: SET[STRING]
 do
   create r.make_from_tuple_array (
     <<["a", 1], ["b", 2], ["c", 3],
        ["a", 4], ["b", 5], ["c", 6],
        ["d", 1], ["e", 2], ["f", 3]>>)
   create ds.make_from_array (<<"a">>)
   -- r is not changed by the query 'domain_subtracted'
   t := r.domain_subtracted (ds)
   Result :=
     t /~ r and not t.domain.has ("a") and r.domain.has ("a")
   check Result end
   -- r is changed by the command 'domain_subtract'
   r.domain_subtract (ds)
   Result :=
     t ~ r and not t.domain.has ("a") and not r.domain.has ("a")
 end
```

① query

② command

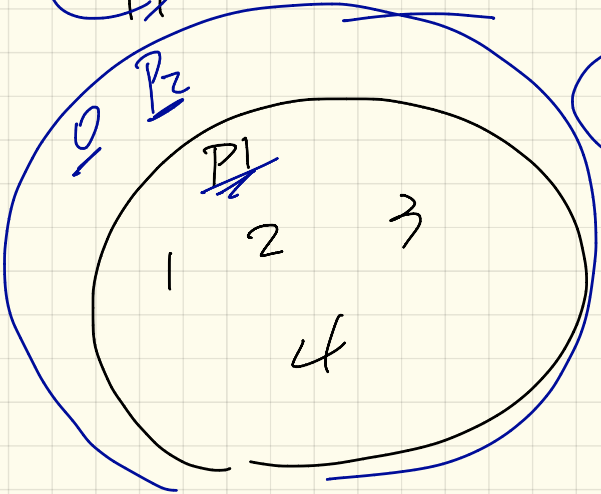
r

Say $r = \{(a,1),(b,2),(c,3),(a,4),(b,5),(c,6),(d,1),(e,2),(f,3)\}$

- r.*domain* : set of first-elements from r
  - r.**domain** = $\{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** = $\{a,b,c,d,e,f\}$
- r.*range* : set of second-elements from r
  - r.**range** = $\{ r \mid (d,r) \in r \}$
  - e.g., r.**range** = $\{1,2,3,4,5,6\}$
- r.*inverse* : a relation like r except elements are in reverse order
  - r.**inverse** = $\{ (r,d) \mid (d,r) \in r \}$
  - e.g., r.**inverse** = $\{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$
- r.*domain_restricted*(ds) : sub-relation of r with domain ds.
  - r.**domain_restricted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \in ds \}$
  - e.g., r.**domain_restricted**({a, b}) = $\{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- r.*domain_subtracted*(ds) : sub-relation of r with domain not ds.
  - r.**domain_subtracted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \notin ds \}$
  - e.g., r.**domain_subtracted**({a, b}) = $\{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- r.*range_restricted*(rs) : sub-relation of r with range rs.
  - r.**range_restricted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \in rs \}$
  - e.g., r.**range_restricted**({1, 2}) = $\{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- r.*range_subtracted*(ds) : sub-relation of r with range not ds.
  - r.**range_subtracted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

P_2 : amount ≥ 0

P_1 : amount > 0



① $P1 \Rightarrow P2$

② $P2 \Rightarrow P1$

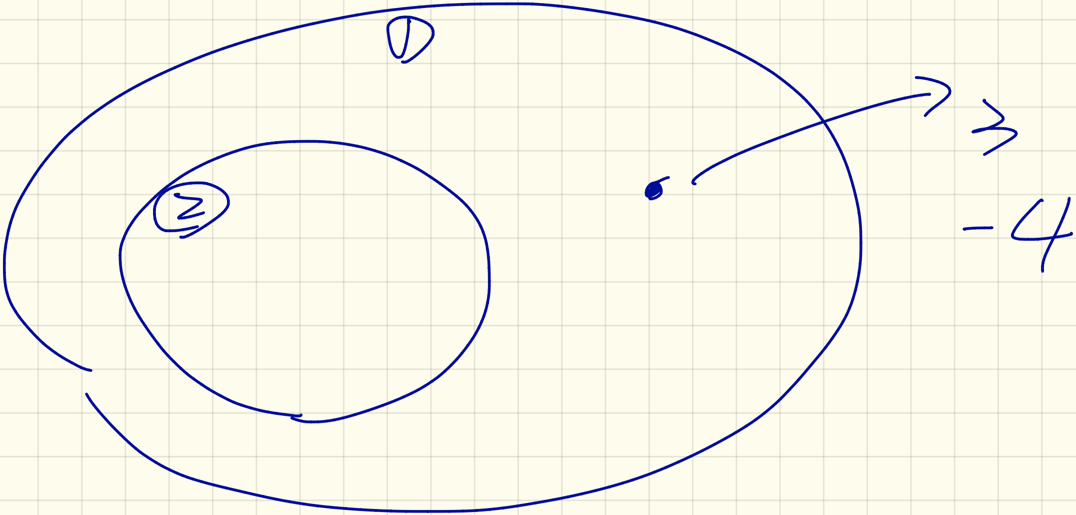
amount = 0 $\quad T \Rightarrow F \quad F$

$q\ (i : \text{INTEGER}) : \text{BOOLEAN}$

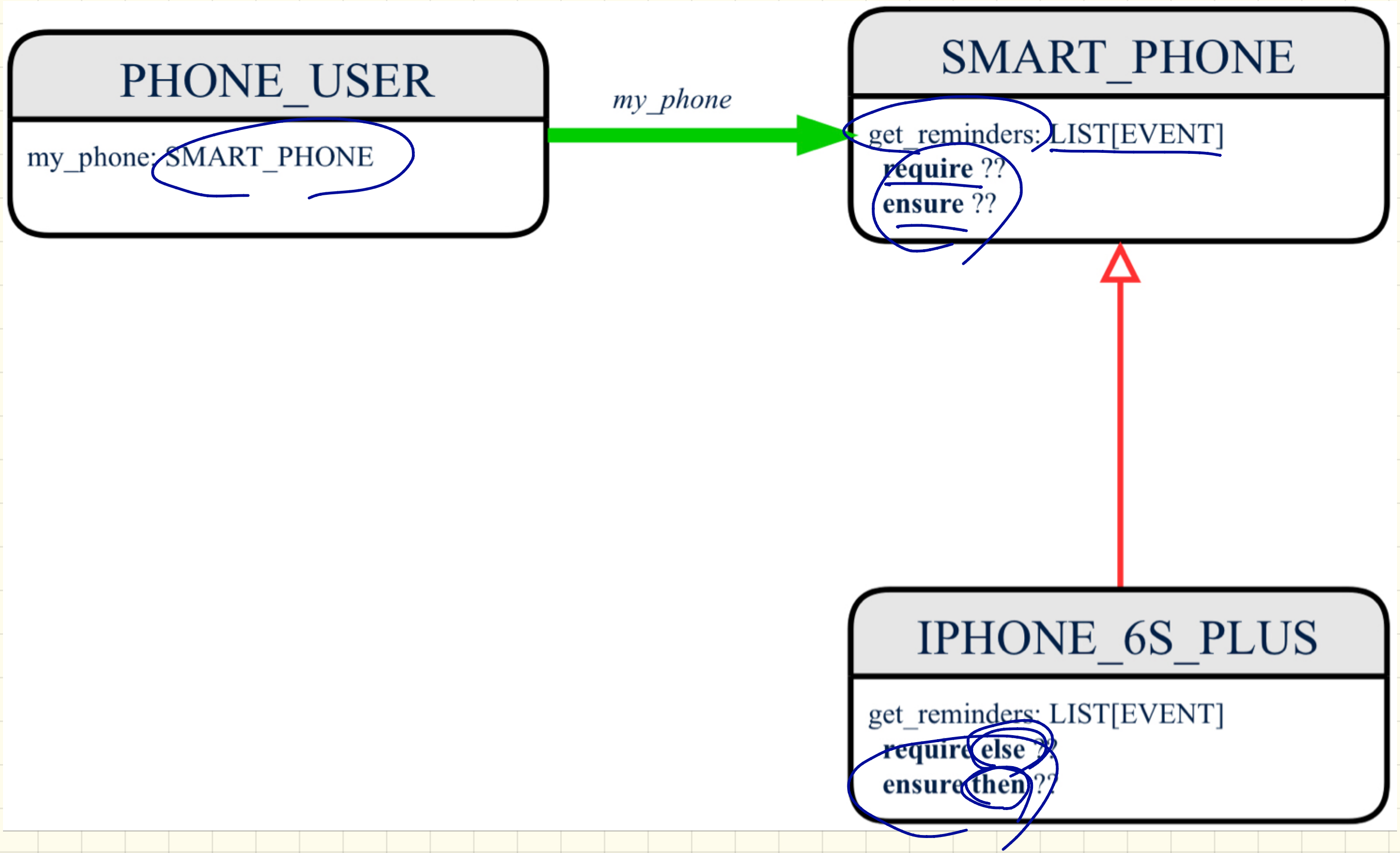
$p \wedge q \Rightarrow p \vee q$

① $\text{Result} = (i > 0) \vee (i \,\%\, 2 = 0)$

② $\text{Result} = (i > 0) \wedge (i \,\%\, 2 = 0)$



Subcontracting : Architectural View



PHONE_USER

my_phone: SMART_PHONE

my_phone

SMART_PHONE

get_reminders: LIST[EVENT]
  require ??
  ensure ??

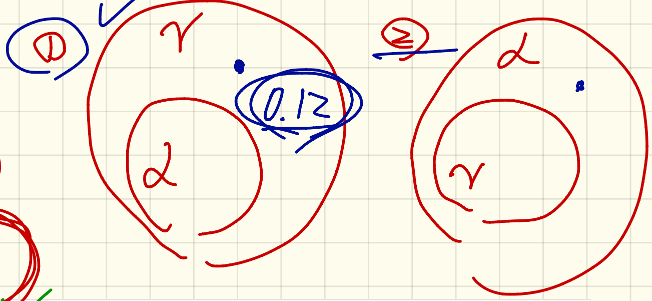
IPHONE_6S_PLUS

get_reminders: LIST[EVENT]
  require else ??
  ensure then ??

Subcontracting : Example (1)

① $\alpha \Rightarrow \gamma$ ✗

② $\gamma \Rightarrow \alpha$ ✓

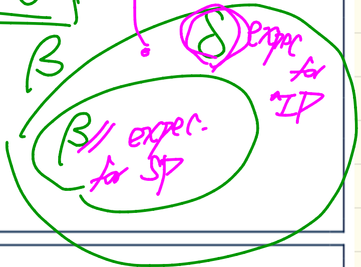


```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

① $\beta \Rightarrow \delta$ ✓

② $\delta \Rightarrow \beta$

e happens tmw



```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```

→ not appropriate ∵ it requires more than α

at100.txt

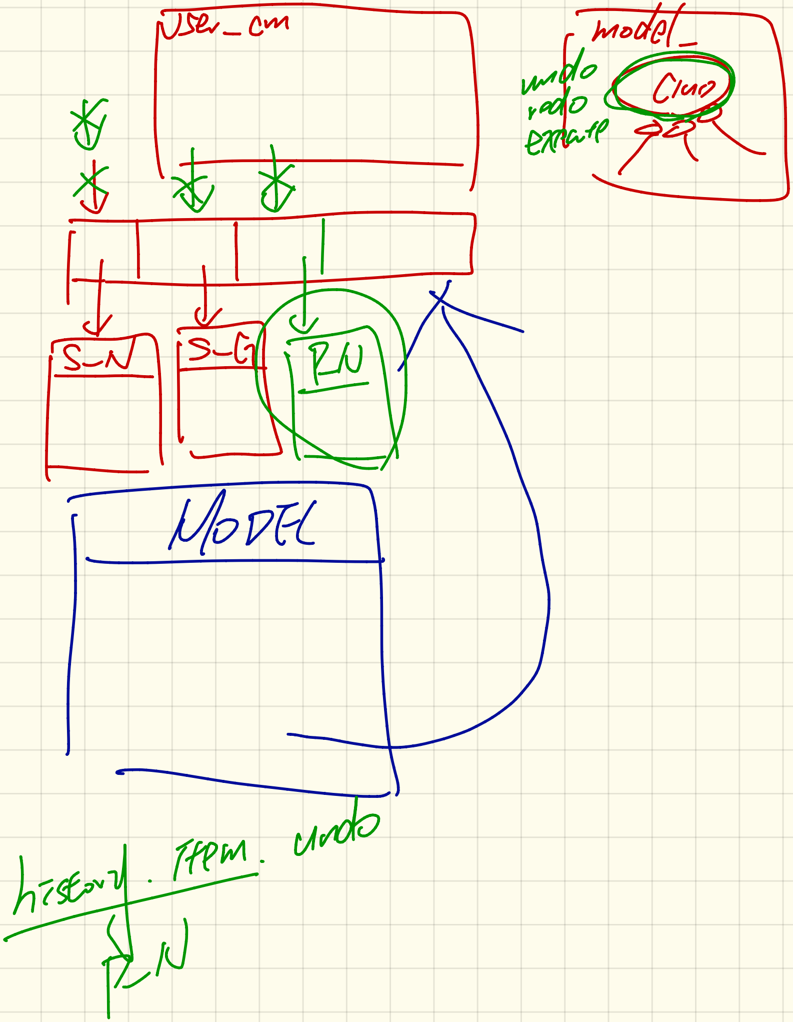
```
set_number(2,1,3)
start_game
put_number(3,4,2)
undo
```

ETF_START_G

start_g

ETF_SET_NUMBER

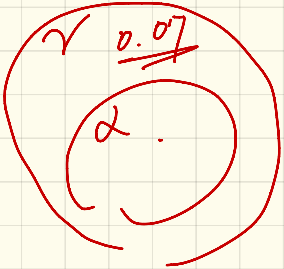
set_number

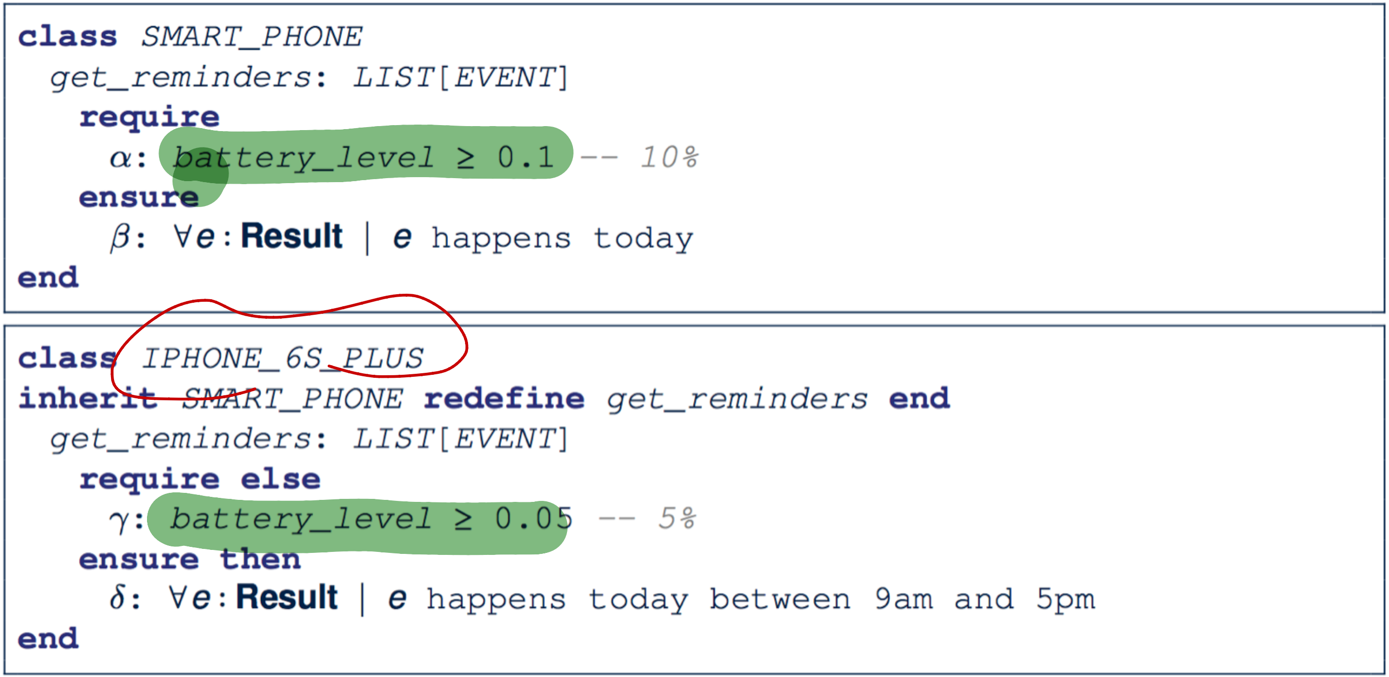


Subcontracting : Example (2)

→ ① $\alpha \Rightarrow \gamma$

② $\gamma \Rightarrow \alpha$





```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e : Result | e happens today between 9am and 5pm
end
```