EECS3311 Software Design (Fall 2020)

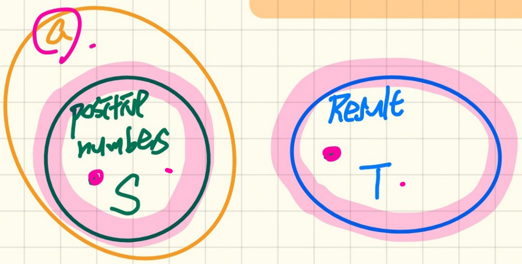
Q&A - Lecture Series W4

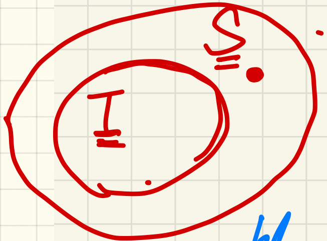
Tuesday, October 5

One-Directional Subset Relation Suffices?

Exercise: Write two postcondition tests.

```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
      no_duplicates: ??
    ensure
      across Result is x
        all
          x > 0
        end
```





$$S = T \equiv (\forall x \mid x \in S \Rightarrow x \in T \quad S \subseteq T$$

$$\land \; x \in T \Rightarrow x \in S) \quad T \subseteq S$$

Exercise
What if we only check $S \subseteq T$ ✓

Having this only is not sufficient

$S = T$

Witness.

T smaller than what it should be.

~~all_pos_in_a_also_in_result :~~
~~across a is x~~
~~all x > 0 implies Result.has(x) end~~

~~all_num_in_result_pos_and_in_a :~~
~~across Result is x~~
~~all x > 0 and a.has(x) end~~

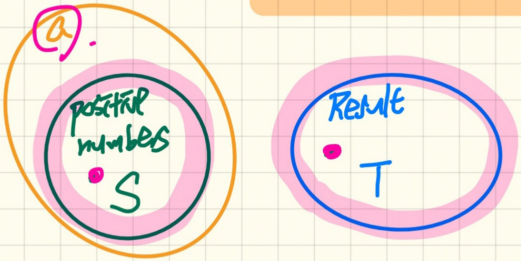
$T \subseteq S$

all_pos_val(<<-1, 2, 3, -2>>)

Result: << 2 >>

wrong ↳ no postcondition violation ↳ wrong output

```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
      no_duplicates: ??
    ensure
①    across Result is x
        all
          x > 0
        end
```



$$S = T \equiv (\forall x \mid \boxed{x \in S \Rightarrow x \in T} \quad S \subseteq T$$

$$\land \ \boxed{x \in T \Rightarrow x \in S}) \quad \boxed{T \subseteq S}$$

✓

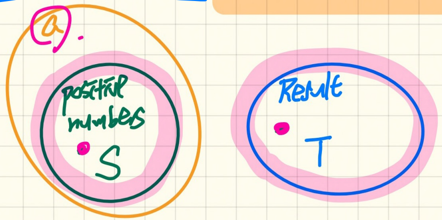
all_pos_in_a_also_in_result:
  across a is x
  all x > 0 implies Result.has(x) end

all_num_in_result_pos_and_in_a:
  across Result is x
  all [illegible] a.has(x) end

②

Complete?

```
all_positive_values(a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
        no_duplicates: ??
    ensure
        across Result is x
            all
                x > 0
            end
```



$S = T \equiv (\forall x \mid x \in S \Rightarrow x \in T$ $\quad S \subseteq T$

$\land \; x \in T \Rightarrow x \in S)$ $\quad T \subseteq S$

all_pos_in_a_also_in_result :
across a is x
all x >0 implies Result.has(x) end

all_num_in_result_pos_and_in_a :
across Result is x
all x>0 and a.has(x) end

a.count = old a.count
and
across 1 |..| a.count is i
all
a[i] ~ (old a.d_t)[i]
end

wrong

a.force(...)

a ~ old E

a.deep_twin

depends on a.object comparison

If we keep the original postcondition, then in the pink postcondition that we are adding is it still necessary to check that every member of Result is positive?

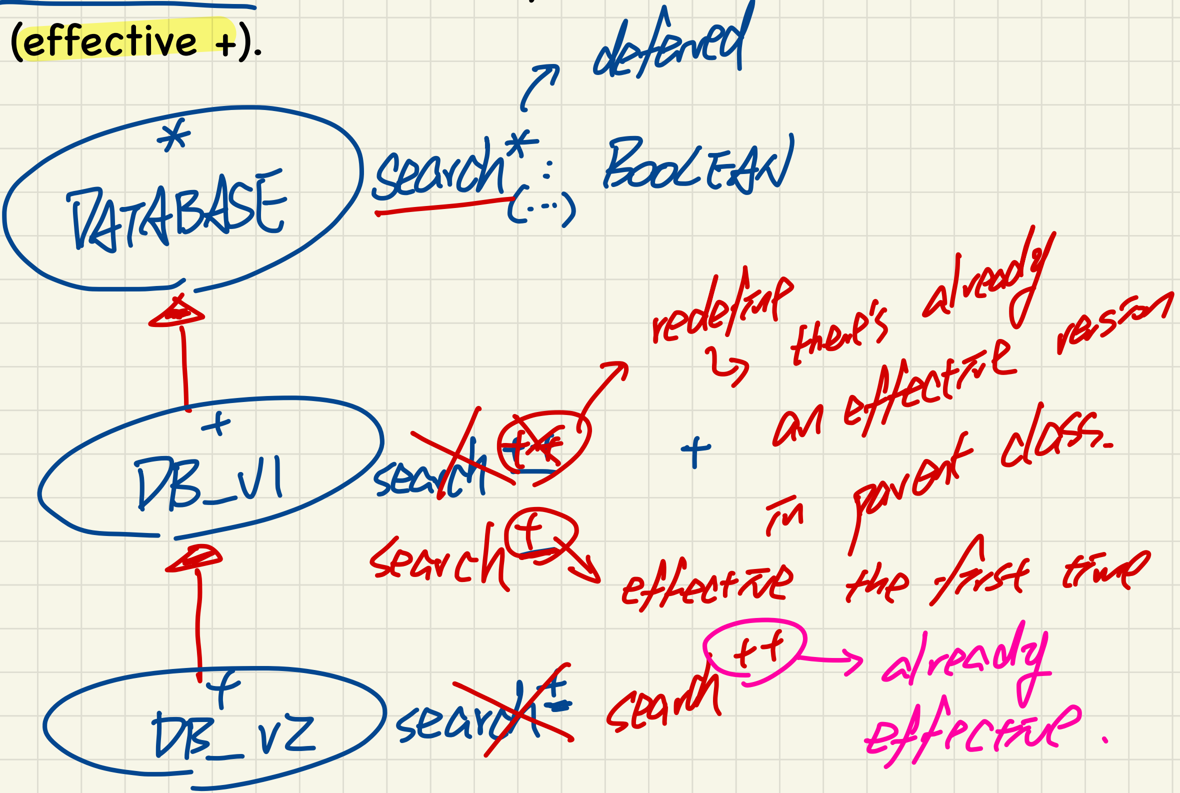
a.obj_comp E.

Also, should we not also check that every member of `a` was uncihanged to fully have a complete postcondition? Something like:

“a.count = (old a.count) and then across 1 |..| a.count is i all a[i] ~ (old a.twin)[i]”

Effective (+) vs. Redefined (++)

when I have a defer routine* (defer class), could I directly (redefine ++) this routine in another class In Eiffel? caz normally after (redefine ++) the routine will become (effective +).



redefine ↳ there's already an effective version in parent class.

effective the first time

already effective.

Clarification of Notation

Could you clarify the set membership; when we are referring to a relation you used a colon to refer to the set membership.

$i : INT$

$i \in INT$

But in this example

"$\forall s : S;\ t1 : T;\ t2 : T \bullet (s, t1) \in f \wedge (s, t2) \in f \Rightarrow t1 = t2$"

you are using the colon to refer to set membership of elements not relations (s is element of S, t1 and t2 are elements of T).

acceptable

To be consistent with the format should it not be:

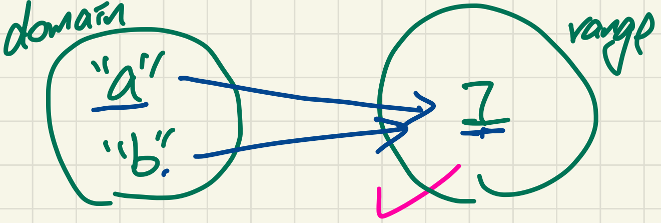
"$\forall s \in S;\ t1 \in T;\ t2 \in T \bullet (s, t1) \in f \wedge (s, t2) \in f \Rightarrow t1 = t2$"

Also rather than semicolon separating the variables is it okay if we use commas (makes it easier to read)?

"$\forall s \in S,\ t1 \in T,\ t2 \in T \bullet (s, t1) \in f \wedge (s, t2) \in f \Rightarrow t1 = t2$"

acceptable.

FUN vs REL



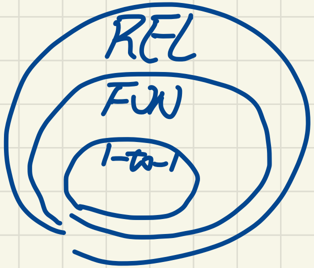
REL ✓

FUN ✓

INJ ✗

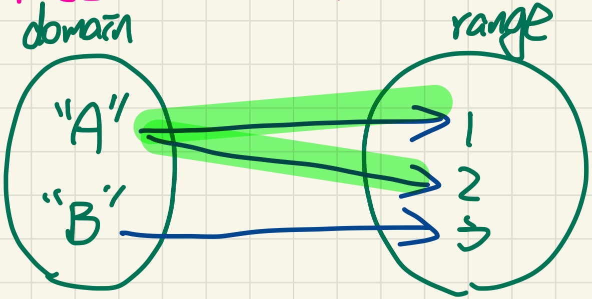
To clarify, REL refers to relationships between sets, and FUN refers to a special case of a ~~set~~? Is that definition correct?

REL



✓ ① $f \in \text{FUN} \Rightarrow f \in \text{REL}$

✗ ② $f \in \text{REL} \Rightarrow f \in \text{FUN}$



valid relation

not a valid function

Exercise

PS $\boxed{8}$

$\{ x, y, z \}$

include/exclude   i/e   i/e

000 $\emptyset$

001 $\{z\}$

010

011

100

101 $\{x, z\}$

110

111

e.g., Say $S = \{1,2,3\}$ and $T = \{a,b\}$, which of the following relations are also functions?

- ○ $S \times T$ diff [No]
- ✓ ○ $(S \times T) - \{(x,y) \mid (x,y) \in S \times T \wedge x = 1\}$ [✓ No]
- ○ $\{(1,a),(2,b),(3,a)\}$ S [Yes]
- ○ $\{(1,a),(2,b)\}$ [Yes]

$\{ \cancel{(1,a)}, \cancel{(1,b)}, \boxed{(2,a), (2,b), (3,a), (3,b)} \}$

$- \{ (1,a), (1,b) \}$

fun? X

Constructing a REL

How to use loops to create tuples in the constructor

(so this is the case when we don't know the elements from before.

So if I have two test cases one can have 2 (value,key) pairs while the other could have 4 (value,key) pairs).

Can you please provide an example?

Starter test in Lab2?

① create Result.make_from_array(
<< ... >>)

② create a.make_empty

[loop] ← go over the DB, and put tuples into a

create Result.make_from_array(a).

③ create Result.make_empty

[Result.extend(_)]

[t , t]

key value.

Use of Commands vs. Queries

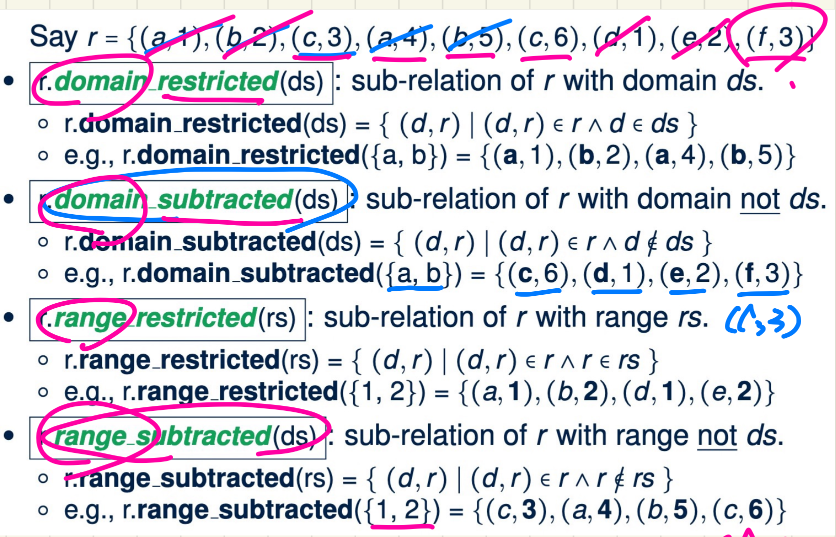
Commands should be used when implementing a model, and Queries should be used when using contracts. Why?

Can't there be instances where one would use commands in queries such as during post-conditions?

```
override_by ( . . . ) [ ]   Command
overriden_by ( . . . ) : like Current ]   query.

ensure .
    model. override_by ( k , v )  X   not compiling .
    model. overriden_by ( . . . ) .  ✓
```

REL Operations



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

- **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. $(c,3)$
  - 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}) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

$(f,3)$

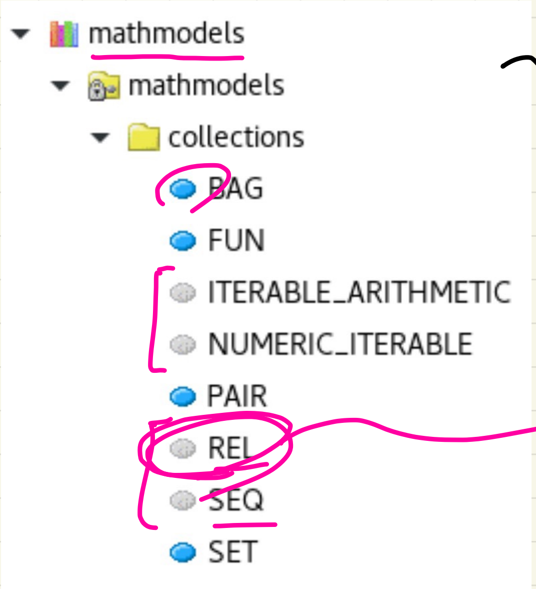
Could you please clarify math domain_subtracted and range_subtracted.

From the example used in the slides, if $\{(d, r) \mid (d, r) \in r \wedge d \notin ds\}$ isn't (c, 3) also supposed to be part of the set for domain_subtracted?

The same goes for range_subtracted. If $\{(d, r) \mid (d, r) \in r \wedge r \notin rs\}$, isn't (f, 3) supposed to be a part of the set?

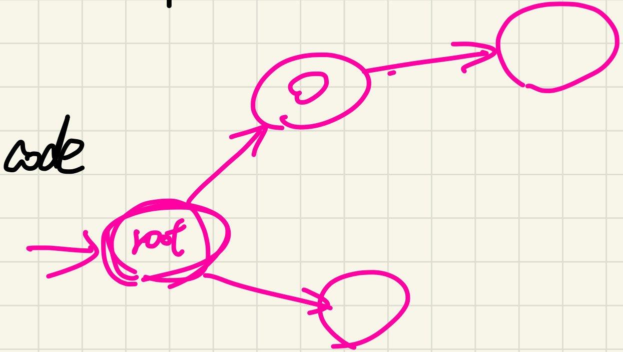
Grey vs. Blue Class Icons in EStudio

Why is the REL class icon in grey? Typically the icon for a class in Eiffel is a blue circle, and if it's deferred there's a red star on top. Same with SEQ and some other classes in mathmodels.



mathmodels
- mathmodels
  - collections
    - BAG
    - FUN
    - ITERABLE_ARITHMETIC
    - NUMERIC_ITERABLE
    - PAIR
    - REL
    - SEQ
    - SET

from the W4 source code



not reachable from the root class.

Missing Invariant BIRTHDAY?

In the Source code provided, I checked the birthday class and I didn't see anything that would stop me from
creating someone's birthday on 31st November(which doesn't exist).

↳ add class invariant.

So Is this a flaw in the implementation or am I missing something?

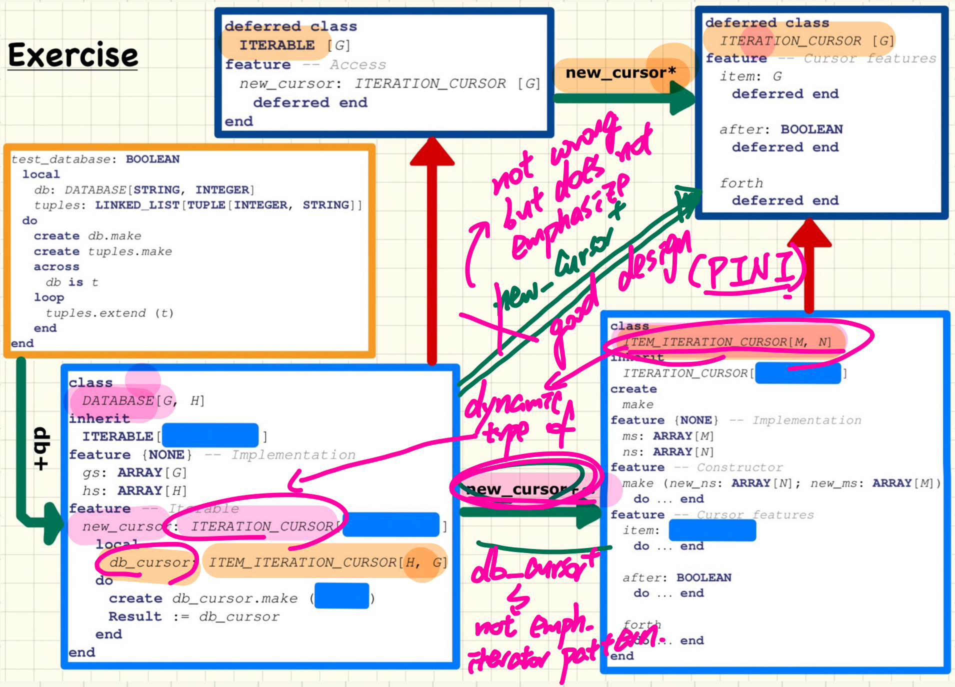
Exercise Solution

Can you please provide the answers to the exercise of abstraction with Trees and Lists/Arrays (How do we convert imp to model when we have trees and also when we have arrays/lists)?

Supplier Type

Could you explain why `DATABASE[G, H]` has a client-supplier relation (in the purple rectangle) with `ITEM_ITERATION_CURSOR[M, N]` rather than `ITERATION CURSOR[G]`?

The new_cursor feature in the database class is of the deferred type iteration_cursor. I understand the + is next to new_cursor because the green arrow is pointing to an effective class, but why is it pointing to it in the first place?



Exercise

```
deferred class
  ITERABLE [G]
feature -- Access
  new_cursor: ITERATION_CURSOR [G]
    deferred end
end
```

```
deferred class
  ITERATION_CURSOR [G]
feature -- Cursor features
  item: G
    deferred end

  after: BOOLEAN
    deferred end

  forth
    deferred end
```

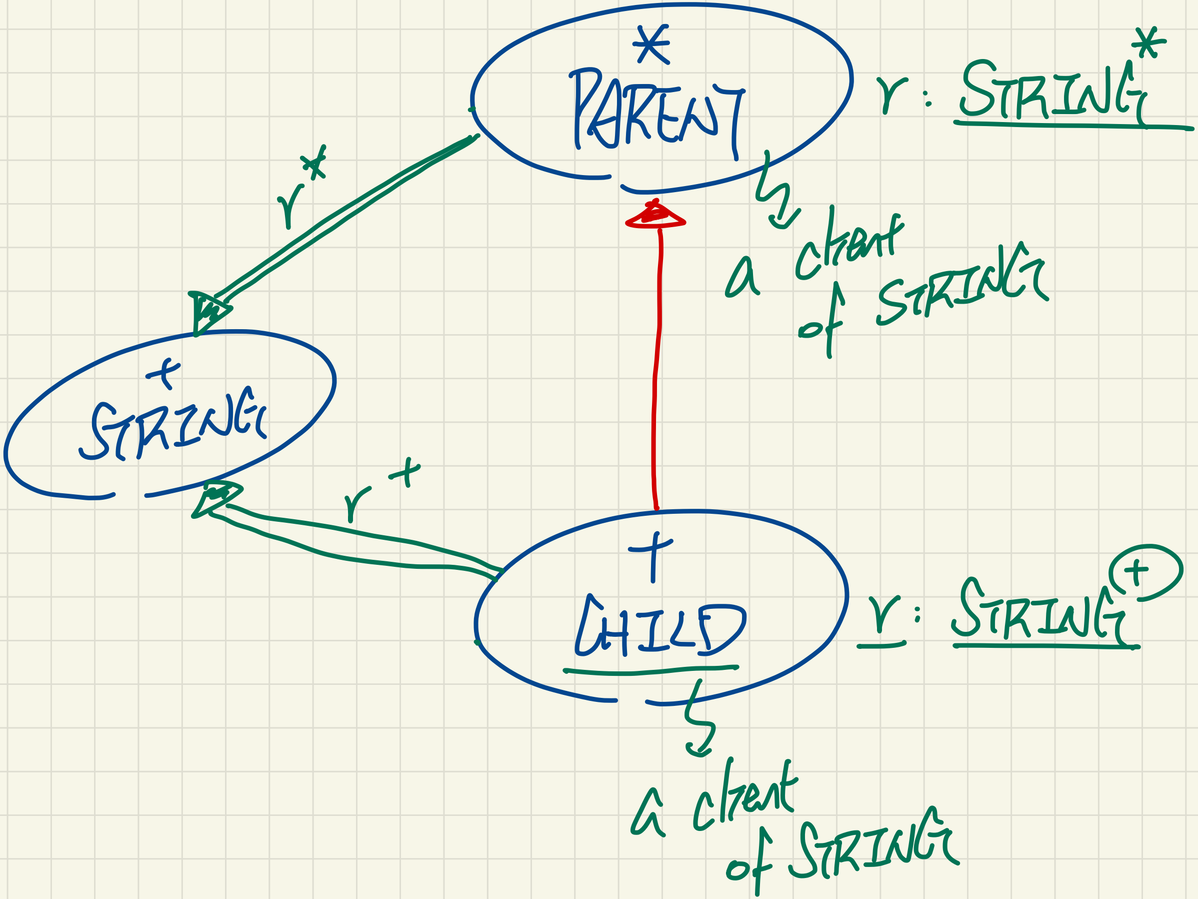
```
test_database: BOOLEAN
  local
    db: DATABASE[STRING, INTEGER]
    tuples: LINKED_LIST[TUPLE[INTEGER, STRING]]
  do
    create db.make
    create tuples.make
    across
      db is t
    loop
      tuples.extend (t)
    end
end
```

```
class
  DATABASE[G, H]
inherit
  ITERABLE[[redacted]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[[redacted]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[H, G]
    do
      create db_cursor.make ([redacted])
      Result := db_cursor
    end
end
```

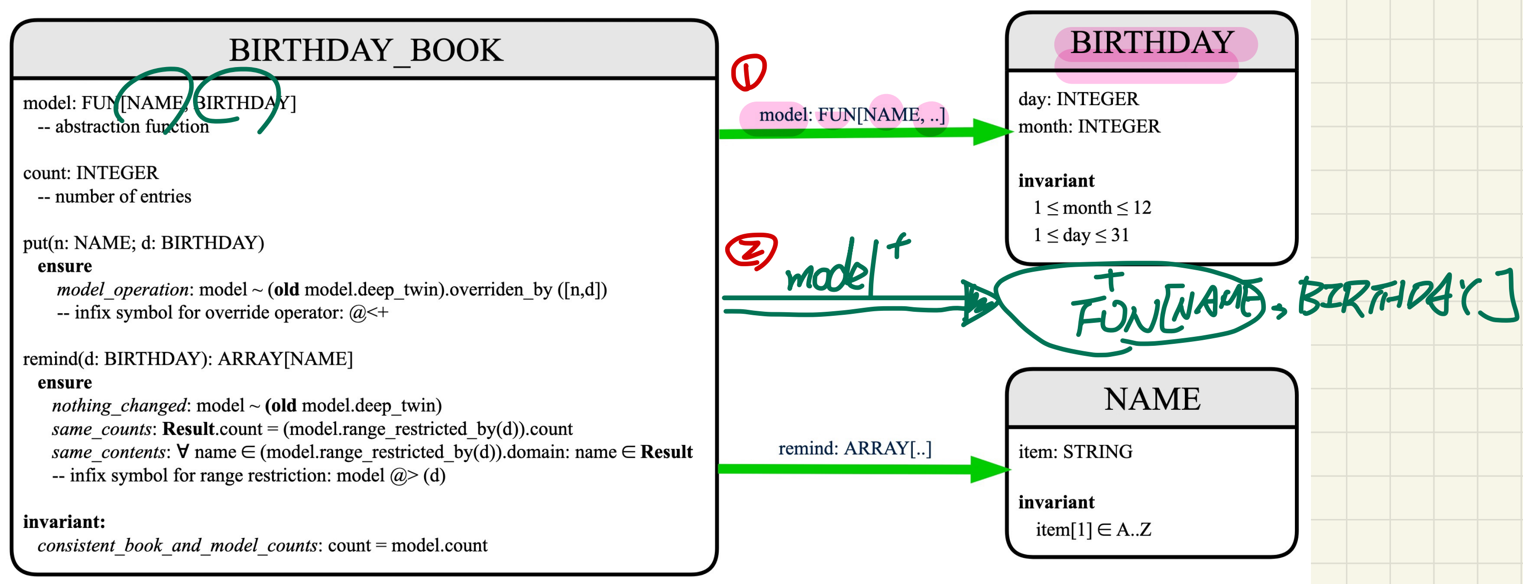
```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[[redacted]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: [redacted]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```


*
PARENT
r : STRING*
a client of STRING
r*
+
STRING
r+
+
CHILD
r : STRING+
a client of STRING

Showing model in Design Diagram



BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]
  -- abstraction function

count: INTEGER
  -- number of entries

put(n: NAME; d: BIRTHDAY)
  ensure
    model_operation: model ~ (**old** model.deep_twin).overriden_by ([n,d])
    -- infix symbol for override operator: @<+

remind(d: BIRTHDAY): ARRAY[NAME]
  ensure
    nothing_changed: model ~ (**old** model.deep_twin)
    same_counts: **Result**.count = (model.range_restricted_by(d)).count
    same_contents: ∀ name ∈ (model.range_restricted_by(d)).domain: name ∈ **Result**
    -- infix symbol for range restriction: model @> (d)

invariant:
  consistent_book_and_model_counts: count = model.count

BIRTHDAY

day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31

NAME

item: STRING

invariant
  item[1] ∈ A..Z

What would this diagram look like if for model, we wanted to emphasize FUN?

Would the arrow labelled "model" point to the FUN class, and would the FUN class then have a client-supplier relation with BIRTHDAY and NAME?

Predicate to Eiffel

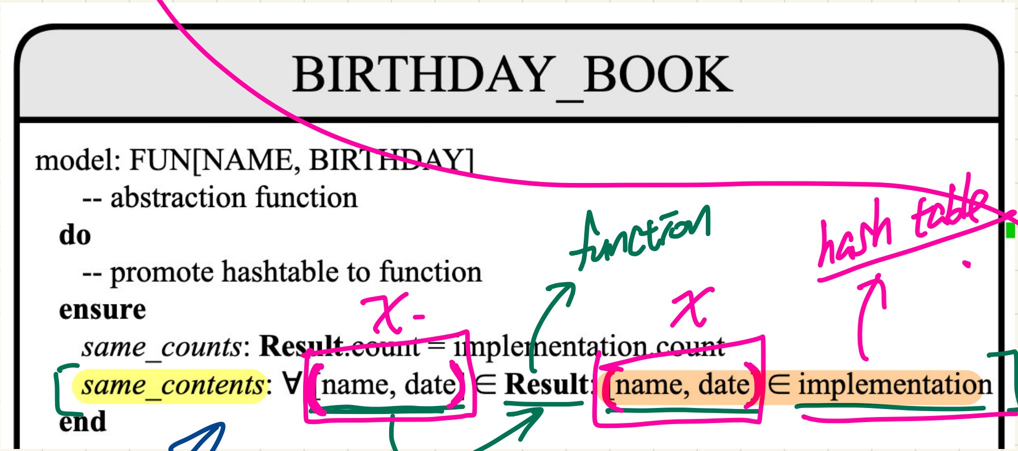
imp.item(x.first) ~ x.second.

fun

Result ⊆ implementation

= hash table

I understand the mathematical representation of the postcondition for same_contents, but I still don't understand how to write it in eiffel?



```
BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]
    -- abstraction function
  do
    -- promote hashtable to function
  ensure
    same_counts: Result.count = implementation.count
    same_contents: ∀[name, date] ∈ Result: [name, date] ∈ implementation
  end
```

[ ] tuple

( ) pair

```
across Result is x
all
    implementation.has_key (x.first)
  and
    implementation.item (x.first) ~ x.second
end
```

x.key × x.first key

x.value × x.second

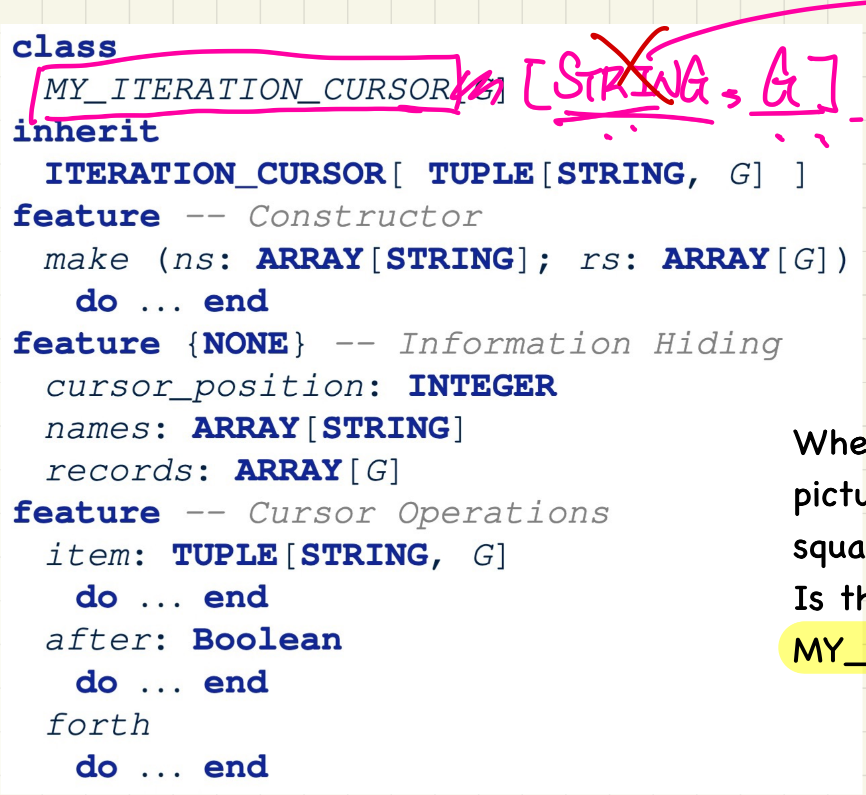
create {PAIR}.make_from_tuple ([2, 3])

[I, I]

[K, V]

PAIR(t) X

Declaring Generic Parameters



```
class
  MY_ITERATION_CURSOR[G]
inherit
  ITERATION_CURSOR[ TUPLE[STRING, G] ]
feature -- Constructor
  make (ns: ARRAY[STRING]; rs: ARRAY[G])
    do ... end
feature {NONE} -- Information Hiding
  cursor_position: INTEGER
  names: ARRAY[STRING]
  records: ARRAY[G]
feature -- Cursor Operations
  item: TUPLE[STRING, G]
    do ... end
  after: Boolean
    do ... end
  forth
    do ... end
```

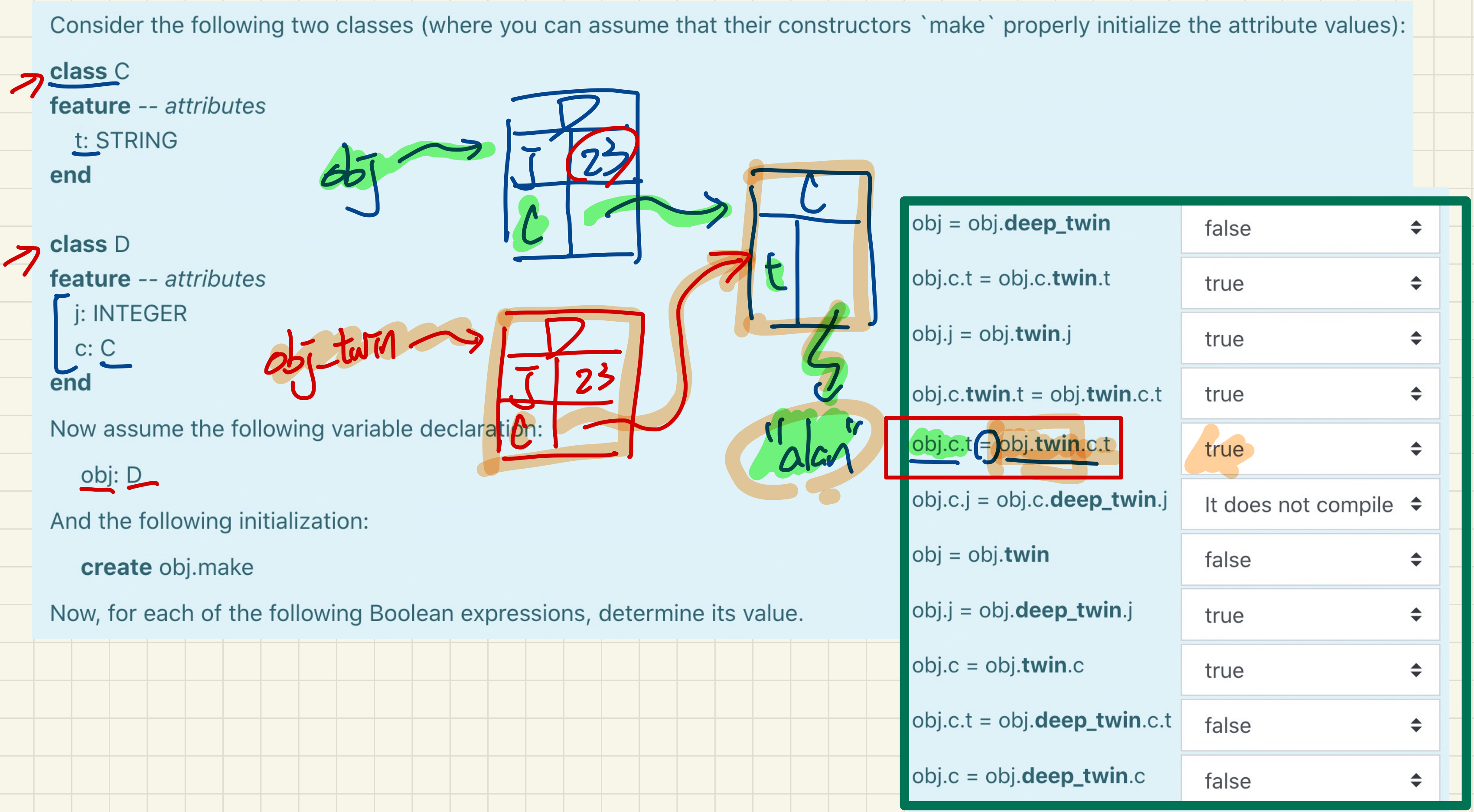
[~~STRING~~, G]

cannot be a valid name to declare a new gen. parameter.

When declaring the class for iteration cursor in the picture below, shouldn't we also have STRING in the square brackets?

Is the following correct: class MY_ITERATION_CURSOR[STRING, G]

Object Copying, old Expressions, Aliasing



Consider the following two classes (where you can assume that their constructors `make` properly initialize the attribute values):

```
class C
feature -- attributes
  t: STRING
end

class D
feature -- attributes
  j: INTEGER
  c: C
end
```

Now assume the following variable declaration:

```
  obj: D
```

And the following initialization:

```
  create obj.make
```

Now, for each of the following Boolean expressions, determine its value.

obj = obj.**deep_twin**	false
obj.c.t = obj.c.**twin**.t	true
obj.j = obj.**twin**.j	true
obj.c.**twin**.t = obj.**twin**.c.t	true
obj.c.t = obj.**twin**.c.t	true
obj.c.j = obj.c.**deep_twin**.j	It does not compile
obj = obj.**twin**	false
obj.j = obj.**deep_twin**.j	true
obj.c = obj.**twin**.c	true
obj.c.t = obj.**deep_twin**.c.t	false
obj.c = obj.**deep_twin**.c	false

```
tn : TREE_NODE[K, V]

① create {PAIR[K, V]}.make(tn.key, tn.value).

② create {PAIR[K, V]}.make_from_tuple(
                                        [tn.key, tn.value]).
```

t

(t.key, t.value)

diagram

(tn.key, tn.value)

ensure .

~~local~~

~~:~~

~~do~~

ensure

① across ______ is □

② attached ______ as □

: