CLEARSY
Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Hackathon
JUL2024

Subject
Block Occupancy

PART I

Thierry Lecomte
R&D Director

THIERRY.LECOMTE@CLEARSY.COM

Objective:
provide a valid modelling on technical subject

≡ Understand the subject and the skeleton of model provided

- It contains a lot of details, it comes from the « real world »
- Be sure that you understand all of it.
- Ask questions if not:
  - live during the sessions,
  - by email the rest of the time (thierry.lecomte@clearsy.com, the subject should start with [HACKATHON]

Objective:
provide a valid modelling on technical subject

≡ Model and verify your modelling

- Top-level modelling questions: set-based specification , substitutions are simple, compact and elegant **(no quantifiers !)**
- Loops to be developed **easily** with abstract iterator
- Check errors: syntax, typecheck
- Generate proof obligations (POs), do automatic proof (force 0 & 1), check remaining POs visually (interactive proof is out of the scope)
- We do not generate C source code but models have to be **B0 checked**
  - Everything deterministic, basic types and table of basic types (including operations parameters and return values), tabular dimensions are constants

Objective: provide a valid modelling on technical subject

≡ Explain and justify

- Why your modelling is correct ? Be concise
- What is your feedback on the subject ? What was easy / tricky ?

Introduction to abstract iterator

- In B, loops are only authorized in implementations, and as such they can only use concrete data and control structures.
- However, the ability to manipulate abstract data (functions, relationships, sets, etc.) is essential in the abstract model and highly desirable in the concrete model.
- The notion of abstract iteration is used as a response to this requirement.

Introduction to abstract iterator

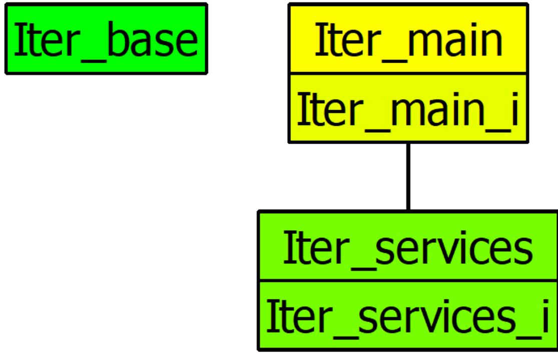
- An abstract iterator is a machine which allows us to walk through a data structure without showing the implementation.
- Iterators do not carry out any treatments other than data examination, and the same iterator can be used in loops carrying out completely different treatments.

B Project Architecture

The ***Iter_main*** machine contains an operations which requires examination of all the elements of a declared set in a given machine (***Iter_base***).

This is carried out by using a loop of ***Iter_main_i*** to iterate for each element of the data.

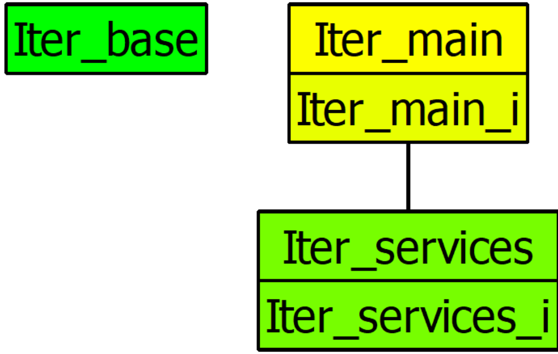
Iter_services provides the abstract iterator



B Project Architecture

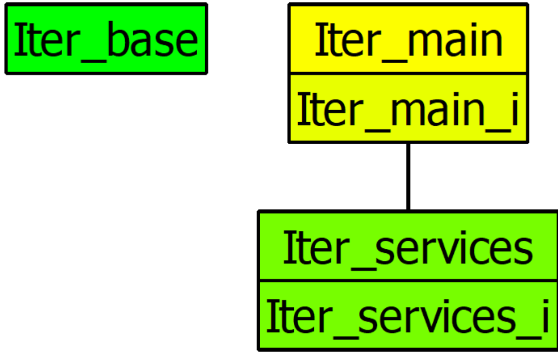
The iterator provides two variables, representing:

- a partition of the set for examination
  - the subset that has already been examined
  - and the subset that still requires examination
- and two operations to initialize the examination and to select an element.



B Project Architecture

The writing of the invariant and the loop is generally simplified, as the difficult aspect - the description of the progress of the calculation procedure - is carried out using abstract data.



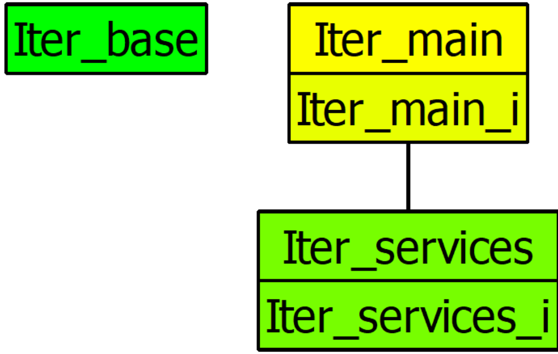
Models

hackathon-2024 / abstract_iterator /

https://github.com/CLEARSY/hackathon-2024

TProver adding asbtract iterator

Name	Last commit message
..	
Iter_base.mch	adding asbtract iterator
Iter_main.mch	adding asbtract iterator
Iter_main_i.imp	adding asbtract iterator
Iter_services.mch	adding asbtract iterator
Iter_services_i.imp	adding asbtract iterator



Iter_base

```
MACHINE
    Iter_base

ABSTRACT_CONSTANTS

    Trains,
    is_MP85

CONCRETE_CONSTANTS

    First, Last

PROPERTIES

    First : NAT &
    Last : NAT &
    First <= Last &
    Last < MAXINT &
    Trains = First..Last &
    is_MP85 : Trains --> BOOL

OPERATIONS

    res <-- is_MP85_op (train) =
    PRE
        train : NAT &
        train : Trains
    THEN
        res := is_MP85 (train)
    END

END
```

Iter_main

```
MACHINE
    Iter_main
SEES
    Iter_base

OPERATIONS

    res <-- nb_MP85 = res := card (is_MP85~[{TRUE}] /\ Trains)
    ;

    res <-- all_MP85 = res := bool (is_MP85~[{TRUE}] = Trains)

END
```

Iter_services

```
MACHINE
    Iter_services
SEES
    Iter_base

ABSTRACT_VARIABLES

    Todo,
    Done

INVARIANT

    Todo <: Trains &
    Done <: Trains &
    Todo \/ Done = Trains &
    Todo /\ Done = {}

INITIALISATION

    Todo := Trains ||
    Done := {}

OPERATIONS

    continue <-- init_iter =
    BEGIN
        Todo := Trains ||
        Done := {} ||
        continue := bool (Trains /= {})
    END
    ;

    continue, elt <-- next_iter =
    PRE
        Todo /= {}
    THEN
        ANY
            chosen,
            new_Todo
        WHERE
            chosen : NAT &
            chosen : Todo &
            new_Todo = Todo - {chosen}
        THEN
            Todo := new_Todo ||
            Done := Done \/ {chosen} ||
            continue := bool (new_Todo /= {}) ||
            elt := chosen
        END
    END
```

Iter_services_i

```
IMPLEMENTATION
    Iter_services_i
REFINES
    Iter_services
SEES
    Iter_base

CONCRETE_VARIABLES

    index

INVARIANT

    index : NAT &
    Todo = index..Last &
    Done = First..(index-1)

INITIALISATION

    index := First

OPERATIONS

    continue <-- init_iter =
    BEGIN
        index := First ;
        continue := bool (index <= Last)
    END
    ;

    continue, elt <-- next_iter =
    BEGIN
        elt := index;
        index := index + 1 ;
        continue := bool (index <= Last)
    END
```

Iter_main_i

```
IMPLEMENTATION
    Iter_main_i
REFINES
    Iter_main
SEES
    Iter_base
IMPORTS
    Iter_services
```

```
OPERATIONS

    res <-- nb_MP85 =
    VAR
        current, continue, current_is_MP85
    IN
        current_is_MP85 := FALSE ;
        res := 0 ;
        continue <-- init_iter ;
        WHILE
            continue = TRUE
        DO
            continue, current <-- next_iter ;
            current_is_MP85 <-- is_MP85_op (current) ;
            IF
                current_is_MP85  = TRUE
            THEN
                res := res + 1
            END
        INVARIANT
            Todo \/ Done = Trains &
            continue = bool (Todo /= {}) &
            res = card (is_MP85~[{TRUE}] /\ Done)
        VARIANT
            card (Todo)
        END
    END
```

Iter_main_i

```
    res <-- all_MP85 =
    VAR
        current, continue, current_is_MP85
    IN
        current_is_MP85 := FALSE ;
        res := TRUE ;
        continue <-- init_iter ;
        WHILE
            continue = TRUE
        DO
            continue, current <-- next_iter ;
            current_is_MP85 <-- is_MP85_op (current)
            /* TO COMPLETE: res := ... */
        INVARIANT
            Todo \/ Done = Trains &
            continue = bool (Todo /= {})
            /* TO COMPLETE: what is the invariant linking res, Done and is_MP85~[{TRUE}] */
        VARIANT
            card (Todo)
        END
    END
```

Final Objective

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Iter_base	OK	OK	2	2	0	OK
Iter_main	OK	OK	1	0	1	OK
Iter_main_i	OK	OK	24	12	12	OK
Iter_services	OK	OK	8	7	1	OK
Iter_services_i	OK	OK	8	7	1	OK

Ranking

Abstract iterator		8
	complete loop (re := ...)	2
	complete invariant	2
	model TC, POG, PR, B0	2
	explanation	2