Software development using B method

Julien Cervelle

LACL - UPEC

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

Introduction

- B method was invented by Jean-Raymond Abrial
- Member of the “Green team” who win the contest of the US department of defense designing Ada language, late 80ies
- Designed the Z specification language based on set theory
- Designed the B language to add the refinement notion and a proof obligation system

Main concepts

- Formal specification of software
  - Data
  - Operations
  - Constraints
- Tools which ensure everything thing written meet the specifications
  - Proof obligations
  - Invariant conservation
  - Post-condition, pre-condition, loop invariant and variant
- Refinement which allows to gradually reach the final software

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

Data structures in B

- Comes from the ZF set theory
- Base elements are sets and integers (and reals though they are not yet implemented)
- Every complex data structures are based on these elements
- Relations are sets of ordered pairs
- Function are relations
- Sequences and arrays are functions

Typing

- Difference with the ZF set theory
  - Integers are not sets
  - ordered pairs are a special construct (and not $\langle a,b\rangle=\{\{a\},\{a,b\}\}$)
- Everything must be properly typed
  - Symbols are not introduced with their types (like in the C language)
  - Typing is done in the logical formula which specifies the symbol
- Location of typing
  - Invariant for state variables
  - Precondition for operation parameter
  - Selection formula for set comprehension $\{x \mid \phi(x)\}$

B abstract machine

- The internal data of the machine is called its state
  - It consists in variables which contains all values needed to perform the machine tasks
- The environment can perform operations on the machine
  - An operation has possibly an input and possibly an output
  - It can consult and modify the variables
- The B abstract machine look like an instance of some class
  - Complete different philosophy from OO software development
  - One software = one machine

B abstract machine component

Symbols

- CONSTANTS
- VARIABLES
- OPERATIONS

Specification

- PROPERTIES
- INVARIANT
- Substitutions

Example

```
MACHINE
  Counter
CONSTANTS
  max_value
PROPERTIES
  max_value∈ℕ₁
VARIABLES
  count
INVARIANT
  count∈ℕ ∧ count ≤ max_value
INITIALISATION
  count:=0
OPERATIONS
  inc = PRE count<max_value
    THEN count:=count+1
    END;
  dec = PRE count>0
    THEN count:=count-1
    END;
  reset = count:=0;
  set(value) = PRE
      value∈ℕ₁ ∧ value ≤ max_value
    THEN count:=value
    END;
  value ← get = value:=count
END
```

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

First order logic and set theory

- Usual first order logic
  - $\Phi \vee \psi$, $\phi \wedge \psi$, $\neg\phi$
  - $\exists x.(\phi)$, $\forall x.(\phi)$
- Sets and ordered pairs
  - (x,y) but better noted $(x \mapsto y)$
  - Set by extension $\{m_1, m_2, \ldots, m_n\}$
  - Set comprehension $\{x|\phi\}$ or generally $\{x,y,\ldots|\phi\}$
- Predicate are carried by sets through set comprehension
  - Predicate PRED(x), true if x meets some condition c
  - Would be the constant PRED=$\{x|x \text{ meets } c\}$

Operations on sets

- Usual operator
  - $E \cup F$, $E \cap F$, $E - F$, $E \times F$
  - $x \in E$, $E \subset F$
- Power set
  - A type can be used as a set (e.g. $F = \mathbb{N} \times \{v\}$)
  - Power set of E is $\mathbb{P}(E)$
  - Set of finite sets of E is $\mathbb{F}(E)$

Relations

- Relations between two sets (or types) E and F are subsets of $E \times F$
  - $E \leftrightarrow F = \{x,y | x \in E \wedge y \in F\} = \{x \mapsto y | x \in E \wedge y \in F\} = \{(x,y) | x \in E \wedge y \in F\}$
- Relations play an important role in B
  - No object notion in B
  - Just some seldom used "record" notion
  - To add an attribute att of type E to some data D…

    …we add a relation $E \leftrightarrow D$ named att to the machine

Example

MACHINE

StudentTable

SETS

Student; Lesson; Teacher

VARIABLES

teaches,follows

INVARIANT

teaches∈Teacher↔Lesson ∧ follows∈Student↔Lesson

INITIALISATION

teaches:∈Teacher↔Lesson || follows:∈Student↔Lesson

END

Partial functions

- Functions are special kind of relations
- Partial function allows at most one y in relation with some x
  - E ⇸ F = {R|R∈E↔F ∧∀x∀y∀z.(x↦y∈R ∧ x↦z∈R ⟹ z=y)}
- Total function requires exactly one y in relation with any x
  - E → F = {R|R∈E⇸F∧∀x.(x∈E⟹∃y.(x↦y∈R))}
- Usual notion of surjectivity, injectivity and bijectivity
  - For total or partial functions
  - Allows to simply specified cardinalities (databases definition)
  - Each has its notation: ↠ ↣ ⤖ ⤀ ⤔ ⤗

Example

MACHINE

StudentTable

SETS

Student; Lesson; Teacher

VARIABLES

has_teacher,follows

INVARIANT

has_teacher∈Lesson⇸Teacher ∧ follows∈Student↔Lesson

INITIALISATION

has_teacher:∈Lesson⇸Teacher || follows:∈Student↔Lesson

END

Operations on relation and functions

- Inverse
- Composition
- Domain and codomain (range)
- Transitive closure
- Image $R[E] = \{y \mid \exists x.(x \in E \wedge x \mapsto y \in R)\}$
- Domain and codomain restrictions, if $R \in E \leftrightarrow F$:
    - $G \triangleleft R = R \cap G \times F$, $G ⩤ R = R - G \times F$
    - $R \triangleright G = R \cap E \times G$, $R ⩥ G = R - E \times G$
- Relation overridding

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

Substitution

- A formal description of how to modify the variables
- Looks like a method of an object
- Is defined using first order logic
- Can be non-deterministic
- Can have parameters and return values

Some substitutions

- x:=y means x becomes equal to y
  - counter := 0, has_teacher(b_method):=johnson
- IF ϕ THEN s1 ELSE s2 END
  - IF param>max_value THEN x:=max_value ELSE x:=param END
- x:(ϕ) means x becomes as ϕ (previous value of x is denoted by x$0)
  - normalize = x:(x<=max_value)
- Can be generalized as x,y,…:(ϕ)
  - normalize = x,max_value:(x<=max_value)
- x:∈E means x becomes any member of E (useful for initialization)

Other substitutions

- ANY, LET allows to use temporary constants
  - With a given value (LET)
  - With a given property (ANY)
- SELECT, CASE, IF allows to chose the substitution depending on conditions
  - From the value of an expression (CASE)
  - Test sequentially (IF)
  - Chosen non-deterministically among true formulas (SELECT)
- PRE is a special substitution to add preconditions to operations

Parallelism

- If multiple variable are to be modified, the substitutions are done in parallel
  - x:=y || y:=x allows to swap the content of variables
- Can be written
  - $x,y:(x=y\$0 \wedge y=x\$0)$
- No sequential operation of loop in B except for the last refinement (called implementation)

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

B method

- The core of B method is to proof in a software the the specification is correct
- The initialization must make the variable verifies the invariant
- Each operation must maintain the invariant
- Each written formula must be sound
  - Writing $f(x)$ generates the proof obligation that $x \in dom(f)$
  - Taking the maximum of E generates $E \neq \emptyset$ and E has an upper bound
- Note that this makes $\wedge$ non commutative
  - $x \in dom(f) \wedge f(x)=12$ is correct but not $f(x)=12 \wedge x \in dom(f)$

B tools

- Software for B method is Atelier-B

  http://www.atelierb.eu/en/telecharger-latelier-b

- It comes with automated tools for proof
  - Predicate prover
  - Mono-lemma prover
  - Mini prover
  - Interactive prover (was people want to avoid)

Substituting logical formula

- The key notion behind proof obligation generation is the weakest precondition
- If S is a substitution and ϕ a formula, [S]ϕ is the weakest formula such that
  - If the variables verifies [S]ϕ and one applies S
  - Then the variables verifies ϕ once S applied
- For instance, [x:=x+1]x=5 is x=4
  - The automated process replaces x by x+1 in x=5 and obtains x+1=5
- $[x{:}{\in}R]x \geqslant 0$ is $R \subseteq \mathbb{N}$
  - The automated process produces $\forall x.(x \in R \Longrightarrow x \geqslant 0)$

Outline

- Introduction
- B abstract machine
- First order logic of set theory
- Substitutions
- Proof obligations
- Refinement

Refinement

- Refinement allows two things
- Data refinement
  - Allows to narrow the state of the machine to go toward the implementation
  - The refinement requires a "gluing invariant" which describe how the original state is computed from the refined state
- Detail refinement
  - Allows to add more details to what is to be done by the machine
  - This refinement can adds variables to the machine
- The two refinement can be mixed up

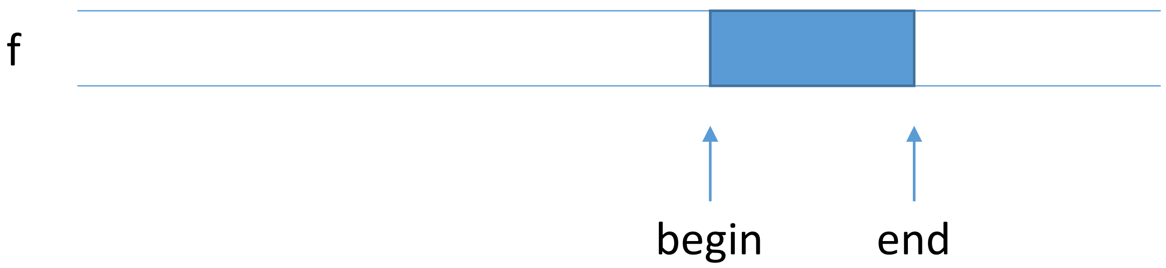
Example: Data refinement

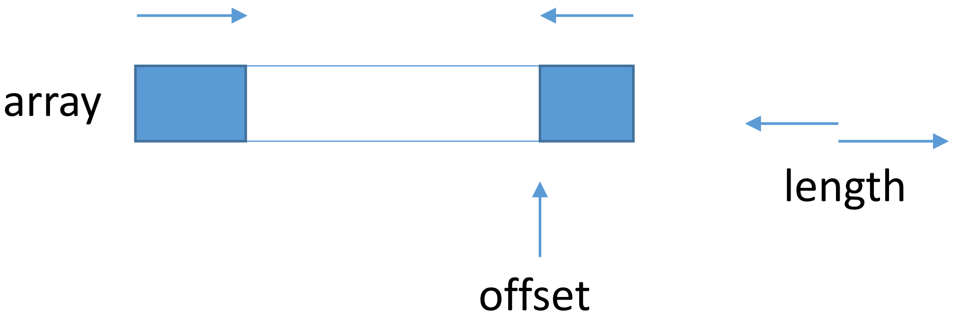
Abstract machine

- Queue of maximum length MAX
- Taking values from a set E
- Represented by:
  - $f \in \mathbb{N} \rightarrow E$
  - begin,end $\in \mathbb{N}$

Concrete machine

- Circular buffer
- Represented by:
  - array: an array of length MAX
  - offset $\in$ 0..MAX-1
  - length$\in$ 0..MAX





Example: Detail refinement

Abstract machine

- Door control system
- status ∈ {OPEN,CLOSE}
- light ∈ {RED, GREEN}
- status=CLOSE ⟺ light=RED

Concrete machine

- open_angle∈0..120 /*degree*/
- light2 ∈ {RED,YELLOW,GREEN}
- open_angle = 0 ⟹ light2 = RED
- open_angle ∈ 1..90 ⟹
  light2 = YELLOW
- open_angle ∈91..120 ⟹
  light2 = GREEN