B Method

Proof assistants

May 16, 2017

Lucas Franceschino

What is B method?

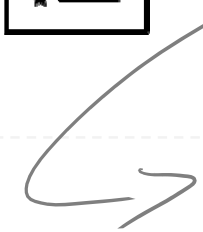
B-method goal

Specifications

Actual program

B-method goal

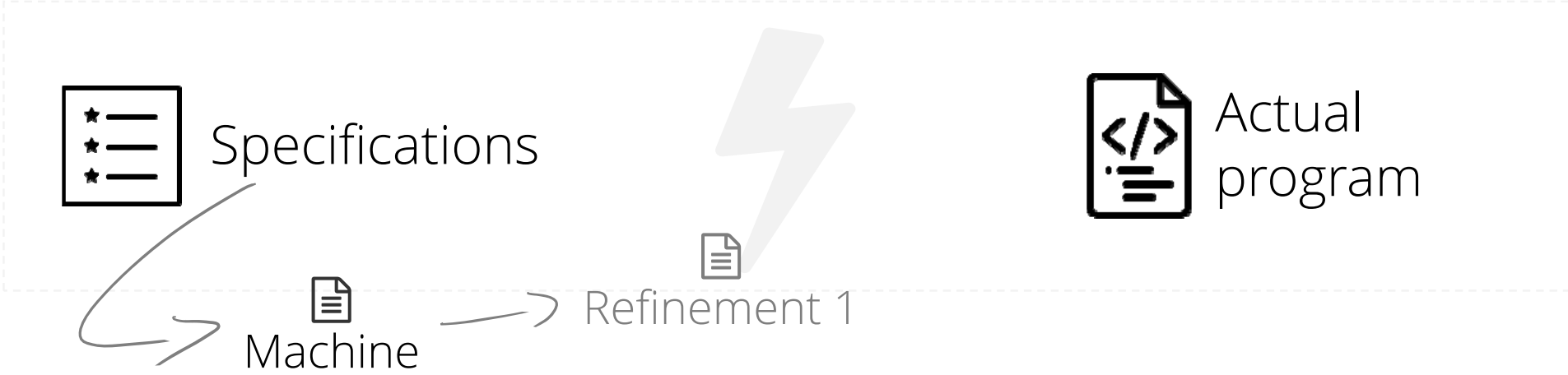
Specifications



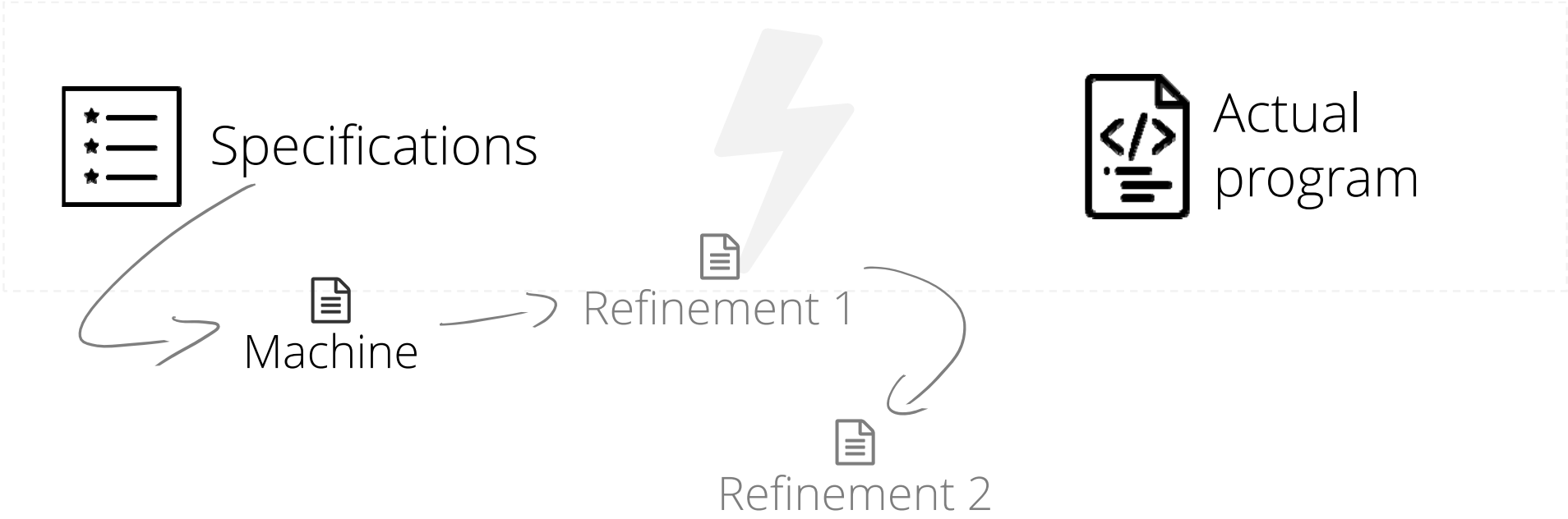
Machine

Actual program

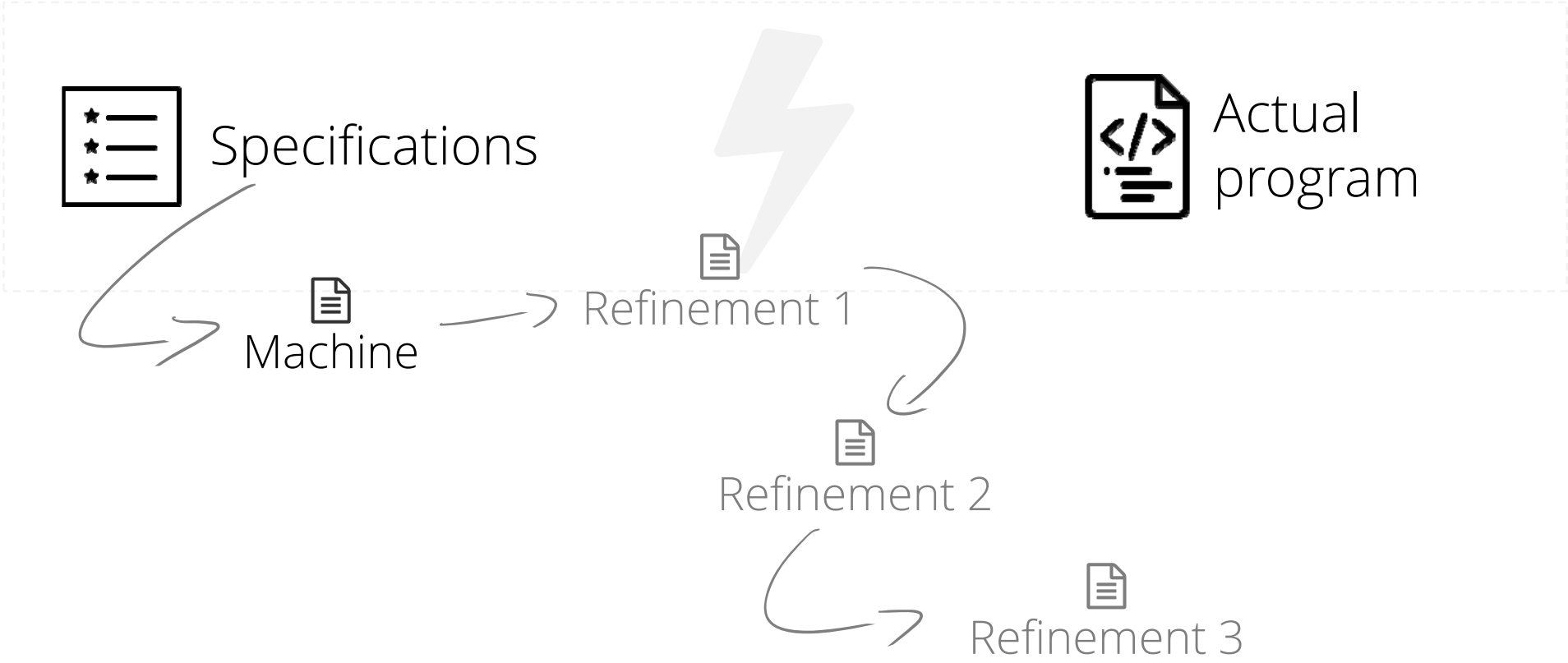
B-method goal



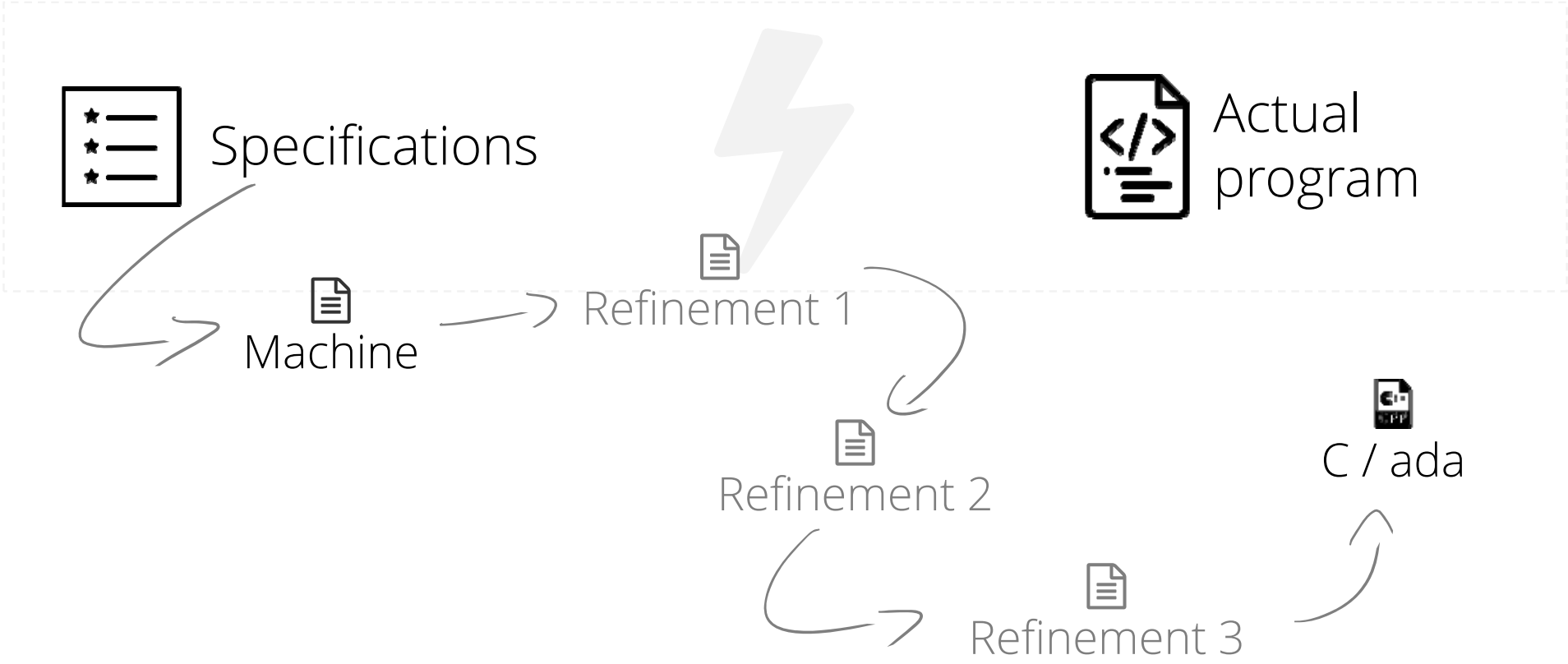
B-method goal



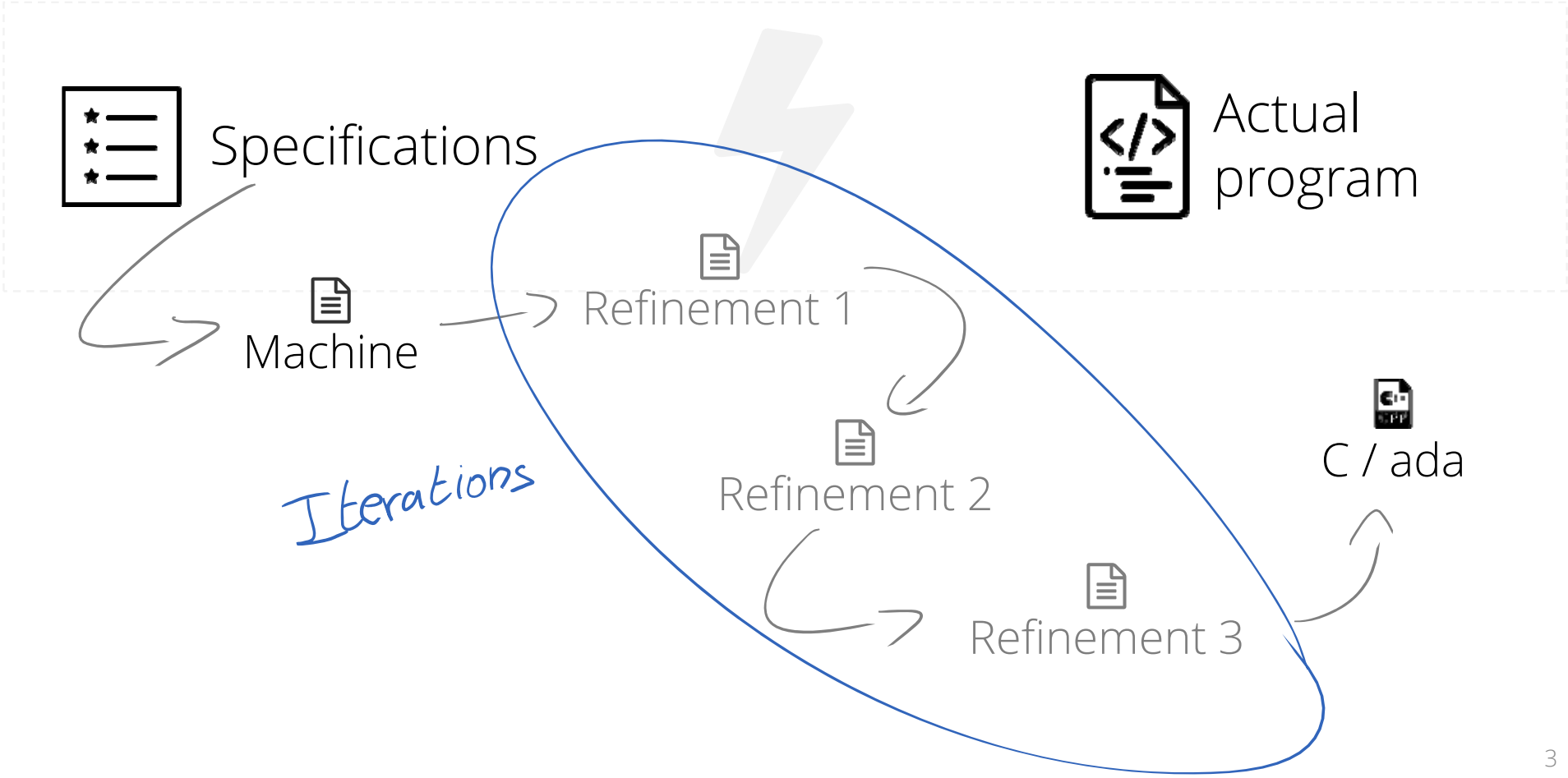
B-method goal



B-method goal



B-method goal



B-method goal

Specifications

Actual program

No gap between **specification** and **actual program**

The initiator of B method

Jean-Raymond Abrial

1970 Specification of data structures and programs

Initiated the Z-Notation (in Oxford)

→ *B4free, Bart, ABTools...*

G. Laffitte,
F. Mejia,
I. Mc Neal

The initiator of B method

Jean-Raymond Abrial

1970 Specification of data structures and programs

Initiated the Z-Notation (in Oxford)

↳ Good for formal specifications, not for development

G. Laffitte,
F. Mejia,
I. Mc Neal

The initiator of B method

Jean-Raymond Abrial

1970 Specification of data structures and programs

Initiated the Z-Notation (in Oxford)

↳ Good for formal specifications, not for development

1996 Published “The B Book”

Atelier B ⟶ *B4free, Bart, ABTools...*

G. Laffitte,
F. Mejia,
I. Mc Neal

The initiator of B method

Jean-Raymond Abrial

1970 Specification of data structures and programs

Initiated the Z-Notation (in Oxford)

↳ Good for formal specifications, not for development

1996 Published “The B Book”

Atelier B ⟶ *B4free, Bart, ABTools...*

2010 Published “Modeling in Event-B : system and software engineering”

Rodin platform

G. Laffitte,
F. Mejia,
I. Mc Neal

The initiator of B method

Jean-Raymond Abrial

1970 Specification of data structures and programs

Initiated the Z-Notation (in Oxford)

↳ Good for formal specifications, not for development

1996 Published “The B Book”

Atelier B → *B4free, Bart, ABTools...*

2010 Published “Modeling in Event-B : system and software engineering”

Rodin platform

↳ Was in the development team of Ada

G. Laffitte,
F. Mejia,
I. Mc Neal

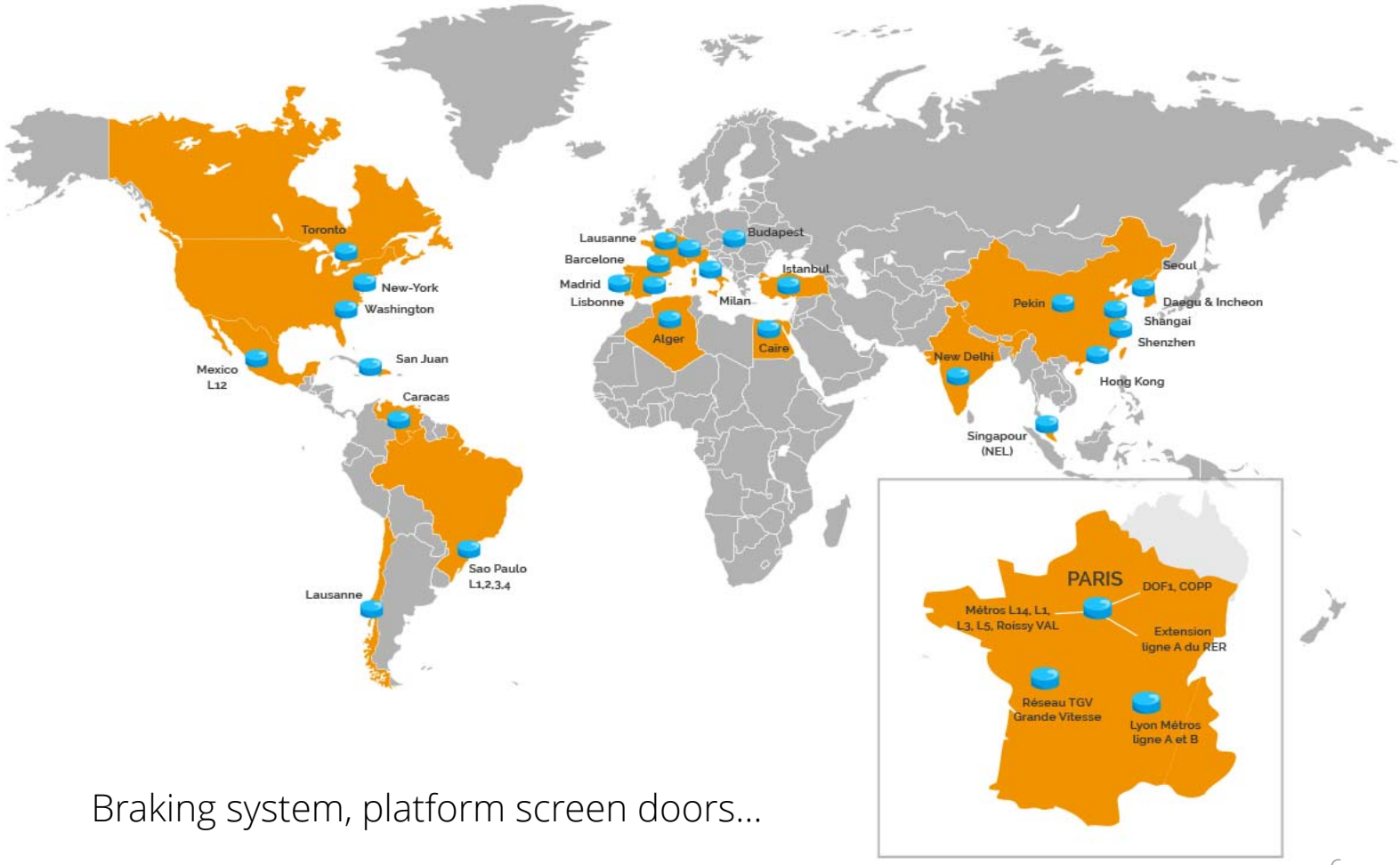
Use cases

Train related B projects around the world

CLEARSY
SYSTEMS ENGINEERING

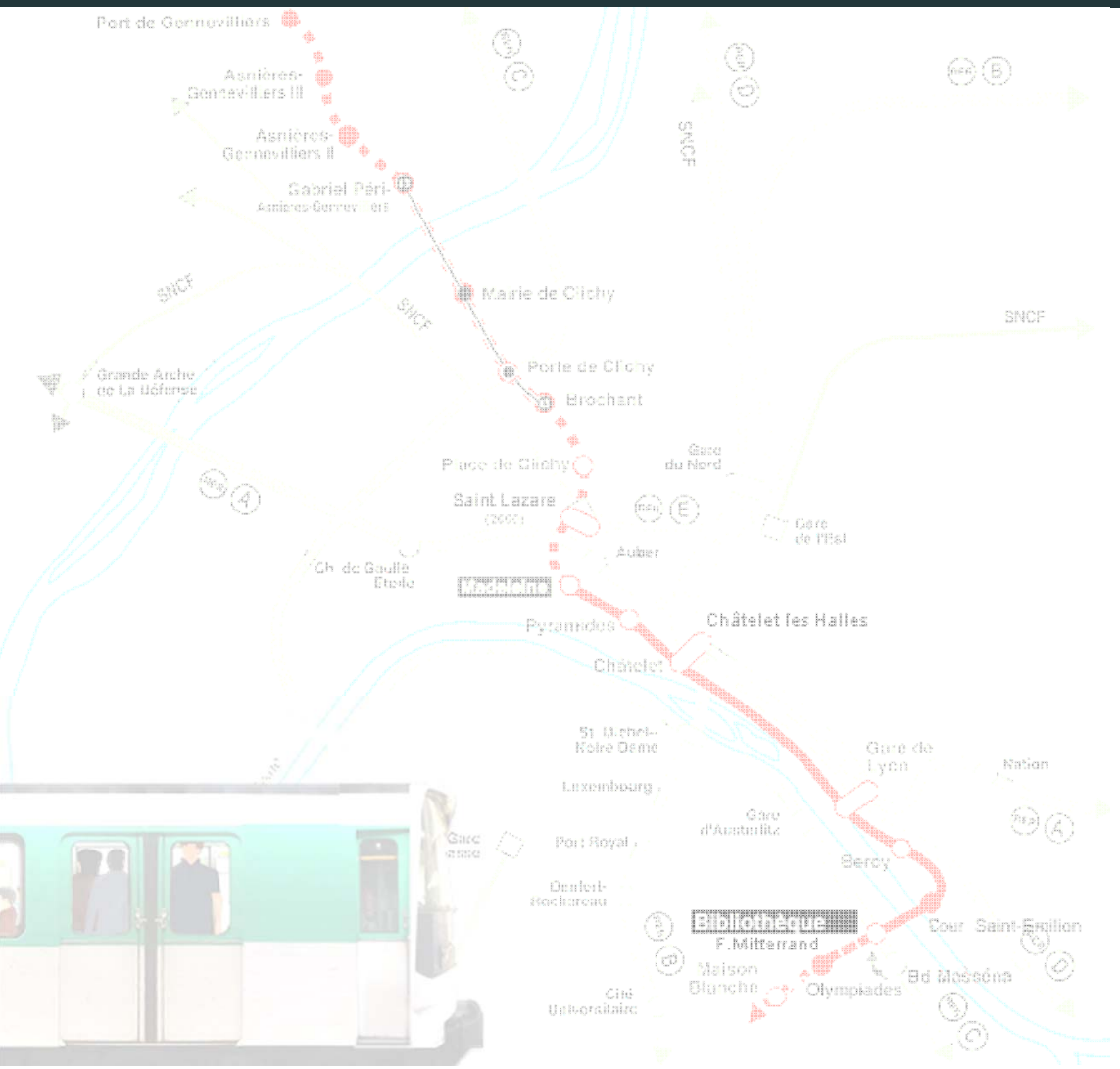
SIEMENS

ALSTOM



Braking system, platform screen doors...

Use case: Meteor in Paris (line 14)

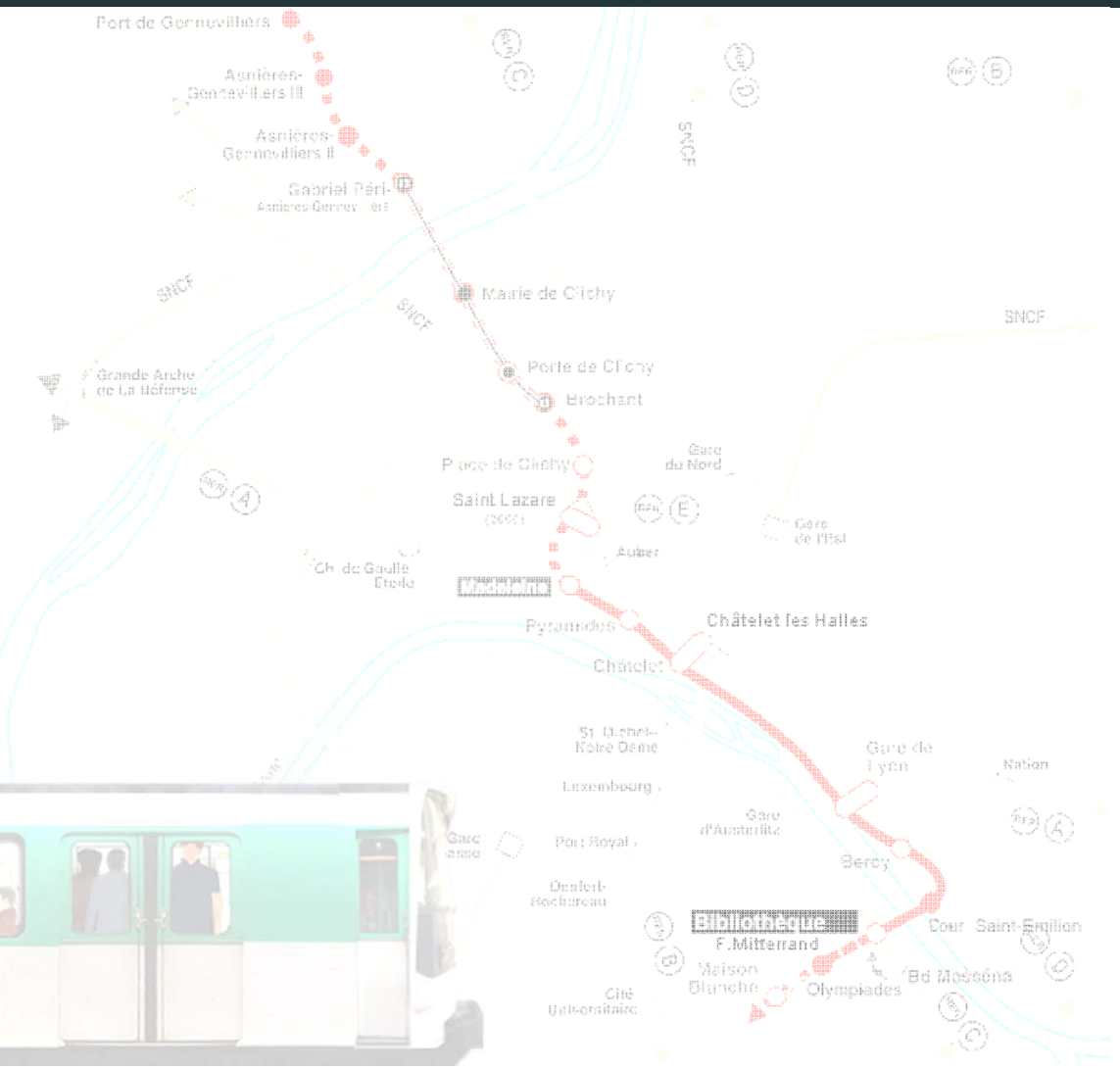


110 000 lines of B </>

29 000 lemmas 👍

87 000 lines of Ada </>

Use case: Meteor in Paris (line 14)



110 000 lines of B </>

29 000 lemmas

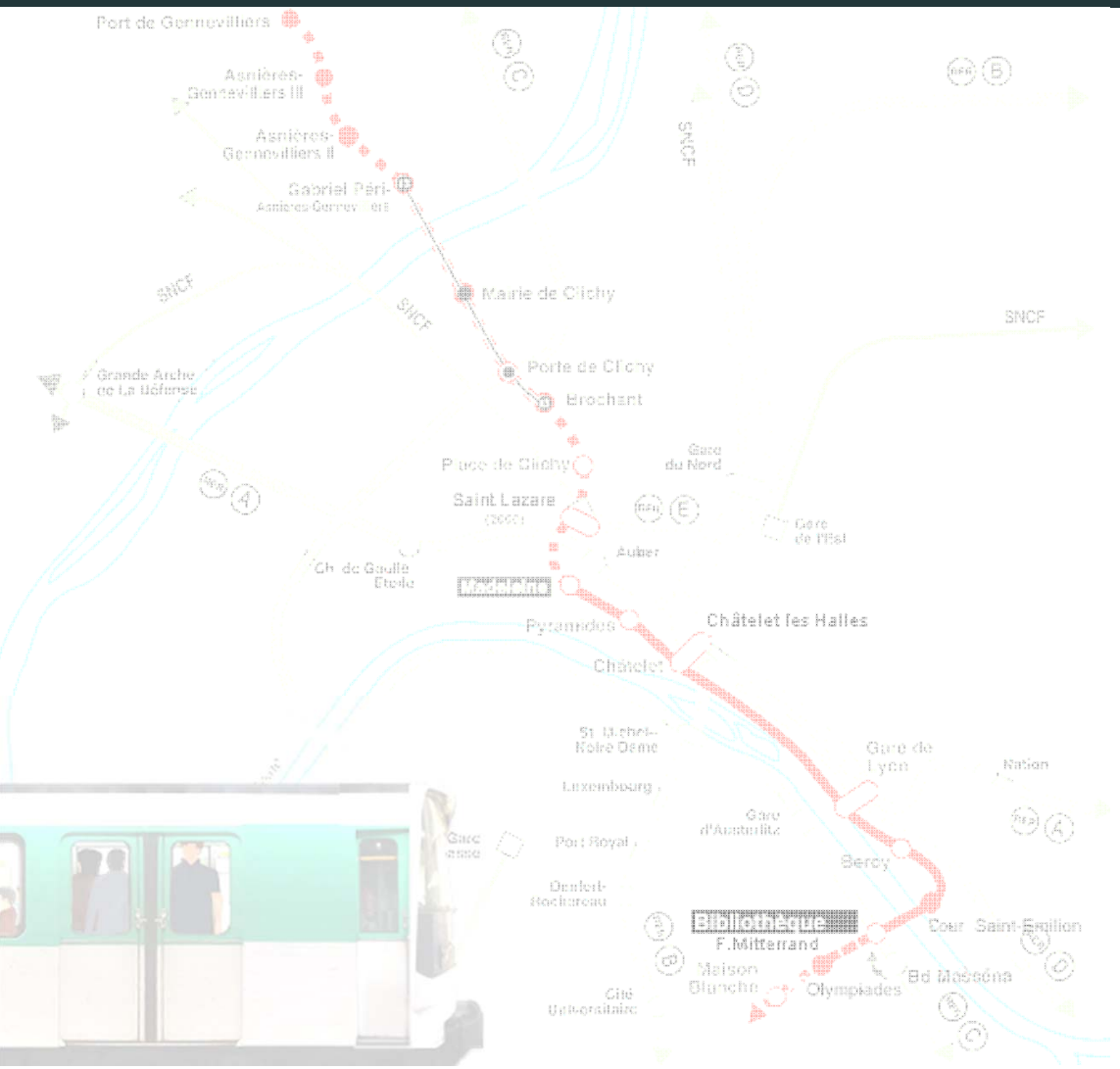
87 000 lines of Ada </>

Driverless trains

Extension in 2003

80 million passengers in 2009

Use case: Meteor in Paris (line 14)



110 000 lines of B </>

29 000 lemmas

87 000 lines of Ada </>

9.2 km

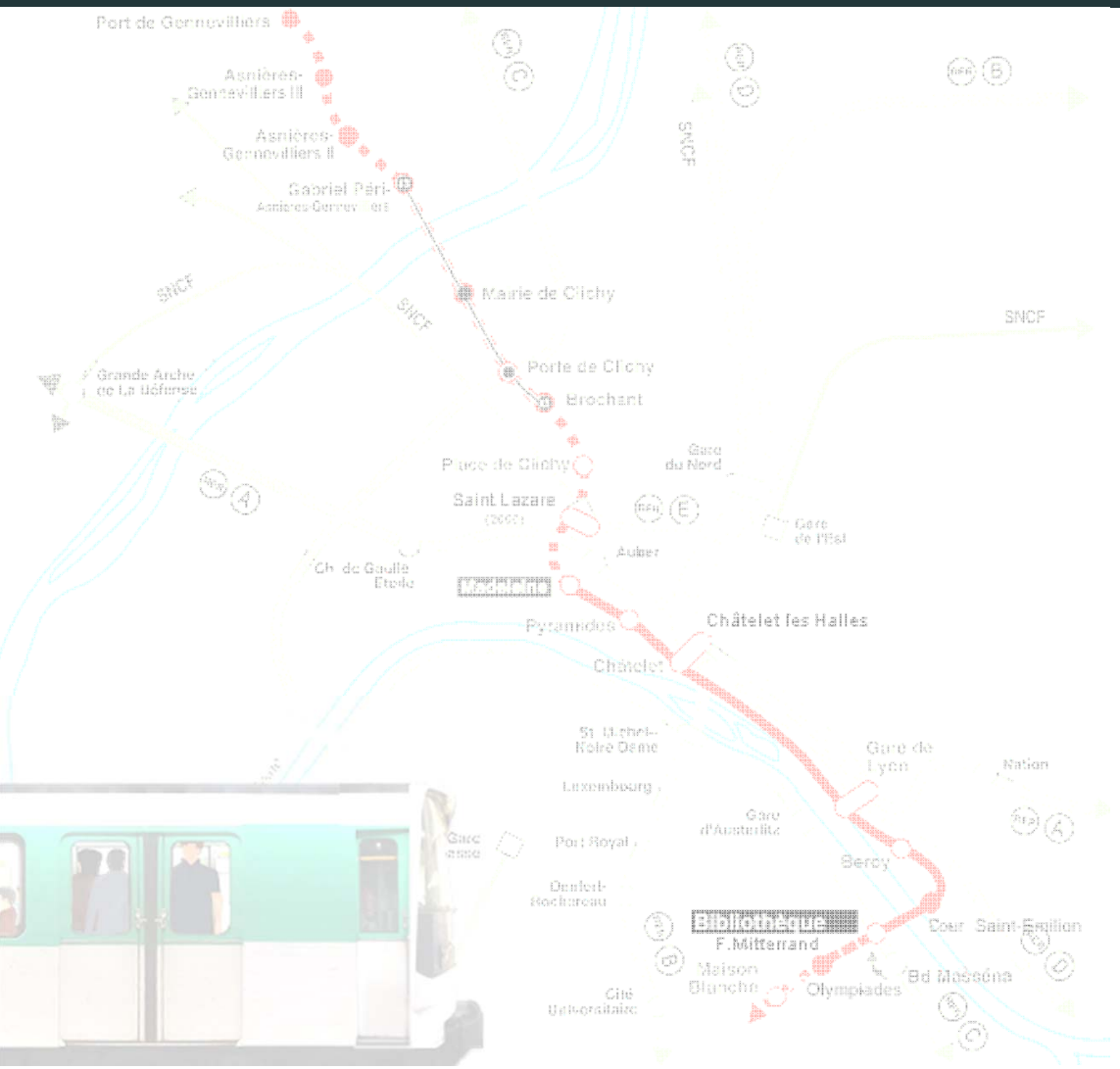
October 1998

Driverless trains

Extension in 2003

80 million passengers in 2009

Use case: Meteor in Paris (line 14)



110 000 lines of B </>

29 000 lemmas

87 000 lines of Ada </>

9.2 km

October 1998

No bugs discovered yet!

Still in version 1.0

Driverless trains

Extension in 2003

80 million passengers in 2009

Other use cases

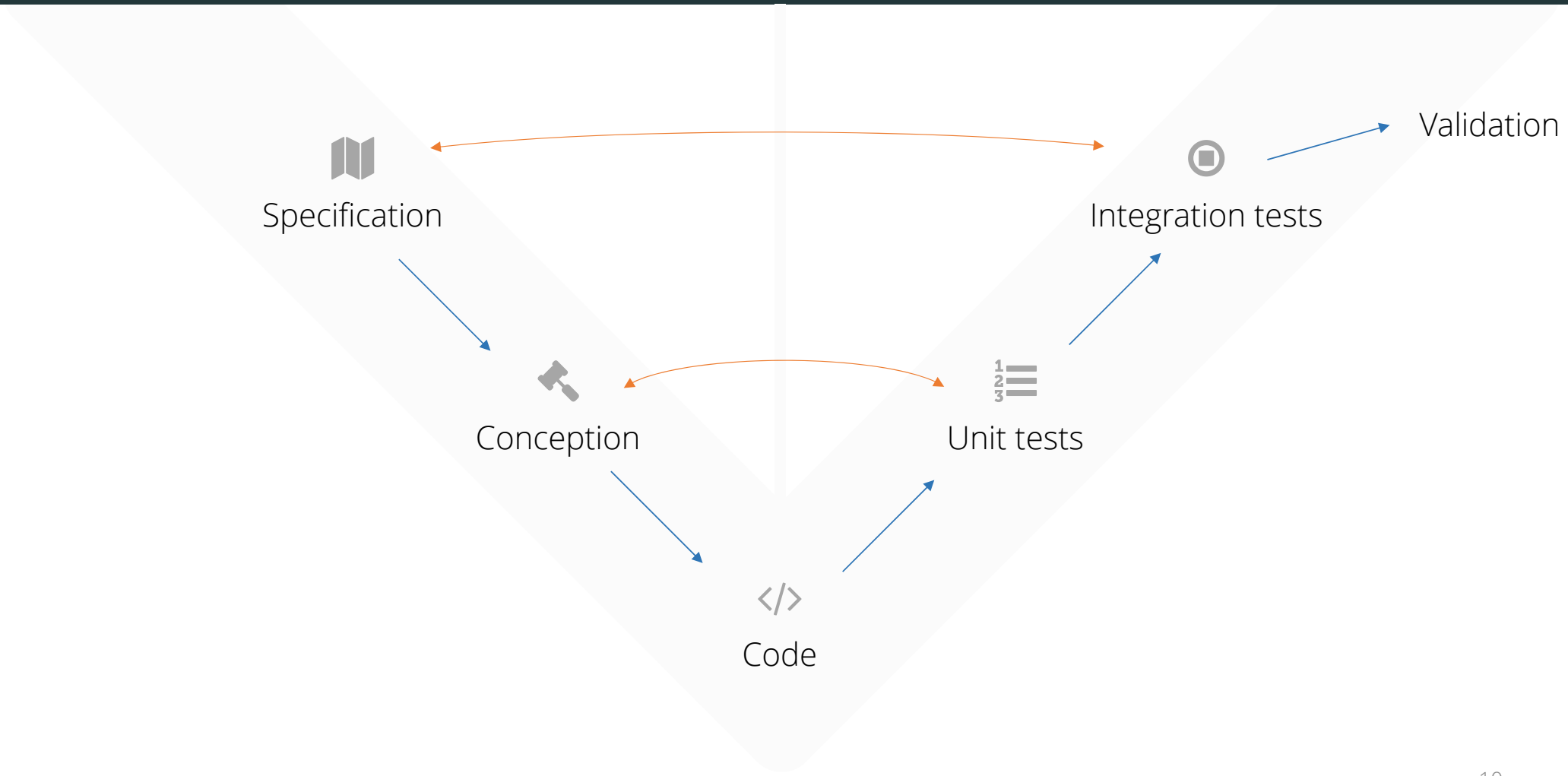
- Peugeot cars: formalization of sub systems (lights system, airbags, motor) to help building diagnostic tools
- Modeling of tasks scheduling from the software controlling the stage separations of Ariane rocket
- Protocol study
- JavaCard runtime formalization

  Java runtime for *smartcard*

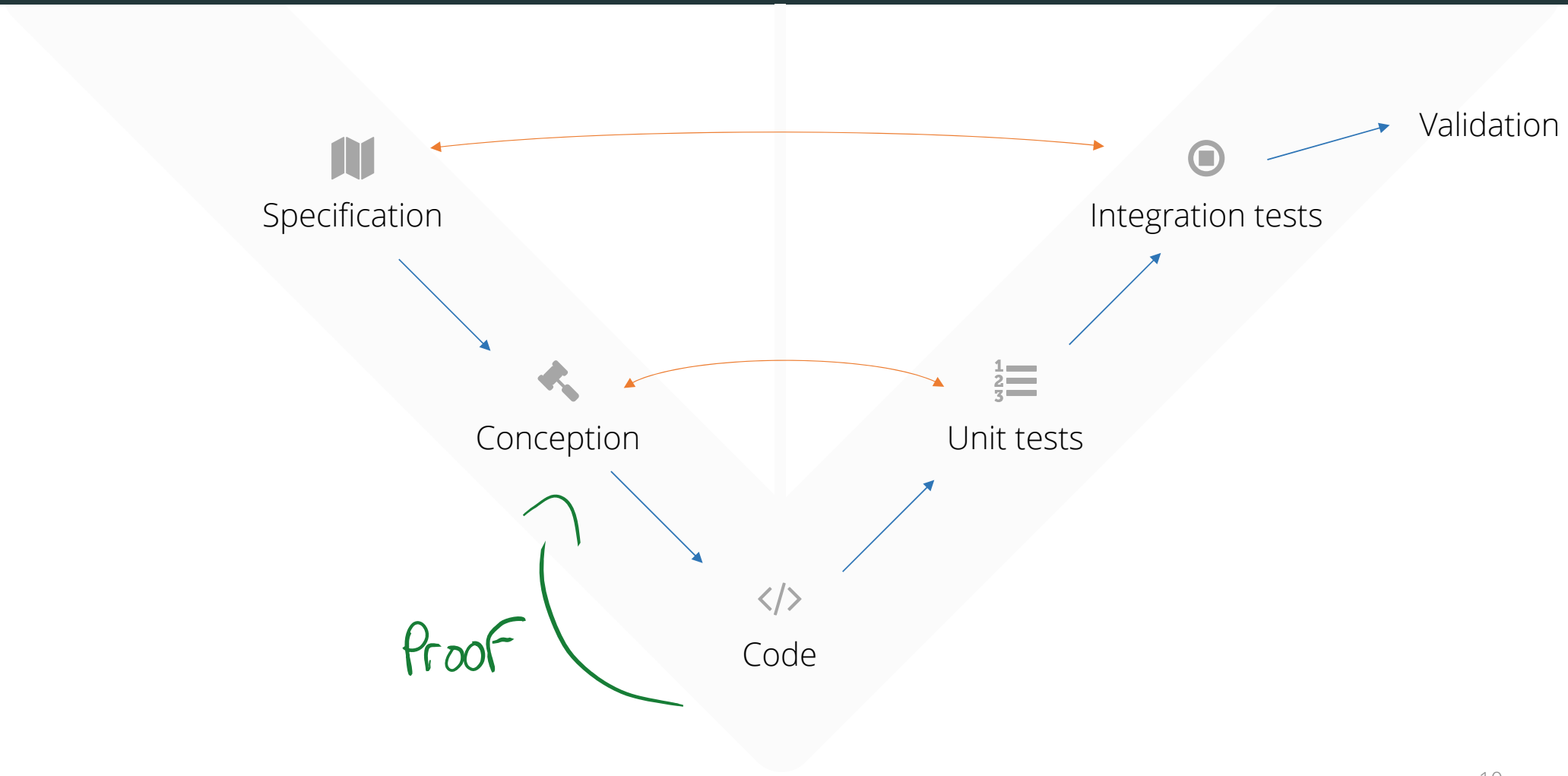
  ↳ Provide *safe:* authentication, data storage, application processing

Developing in B

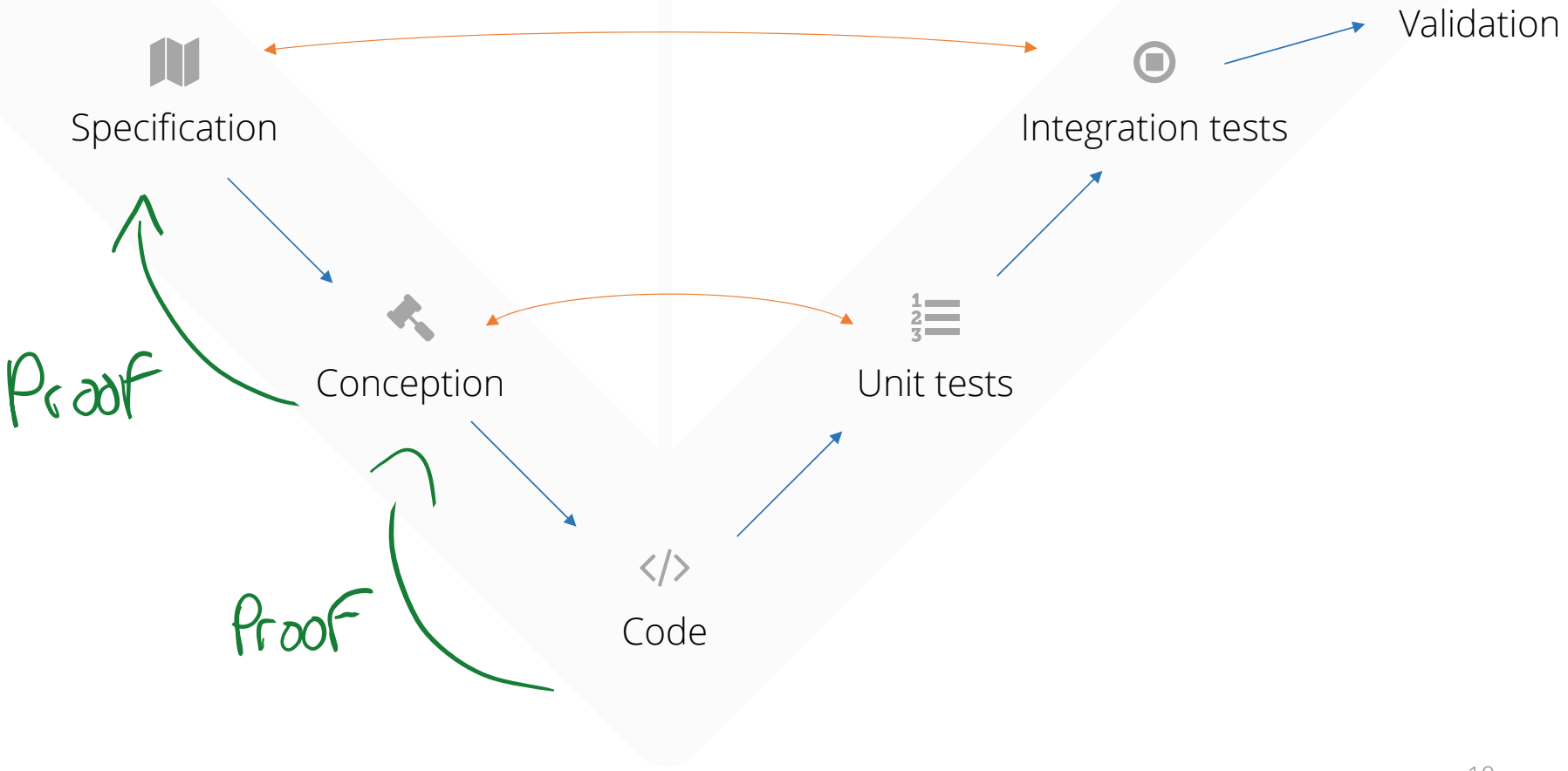
B method development



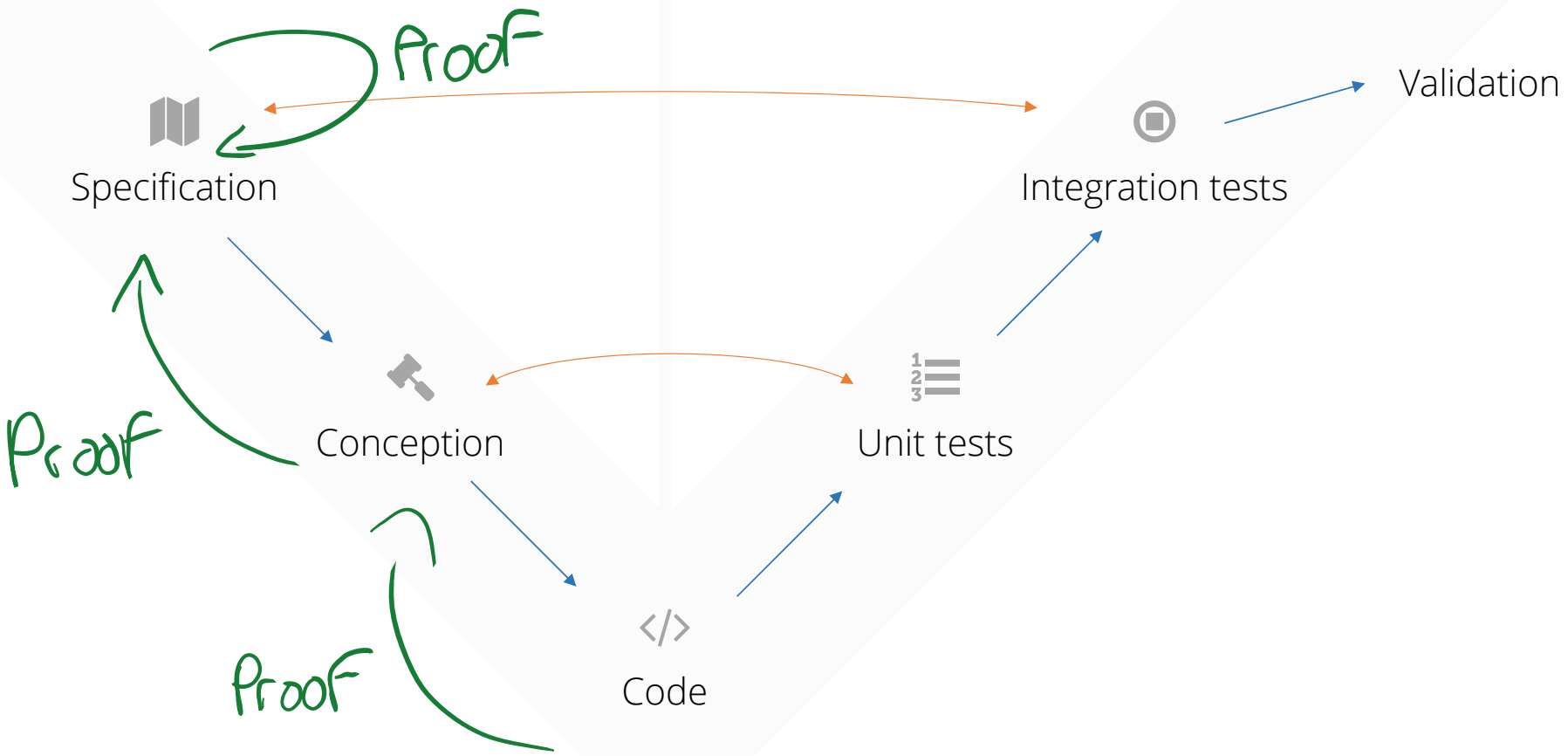
B method development



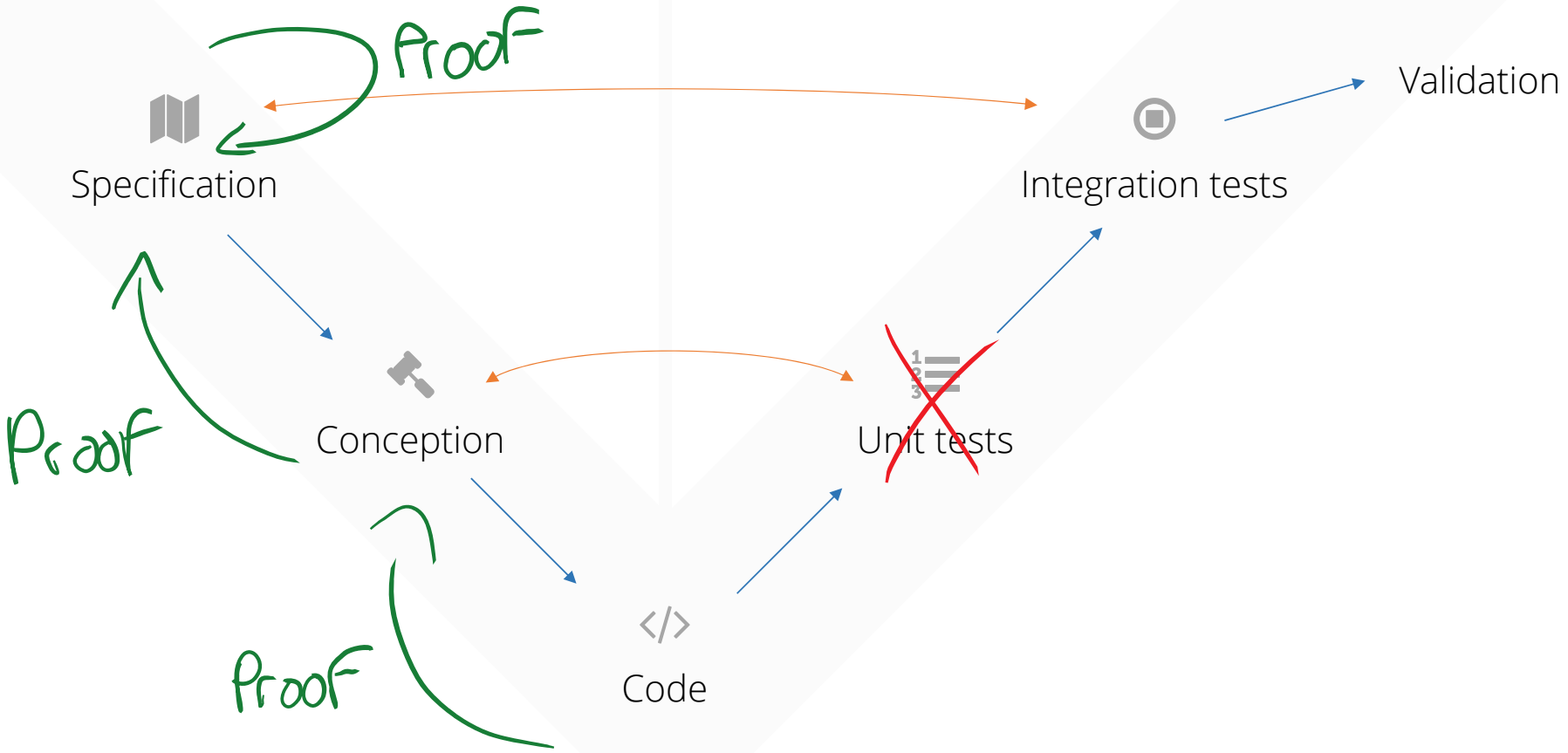
B method development



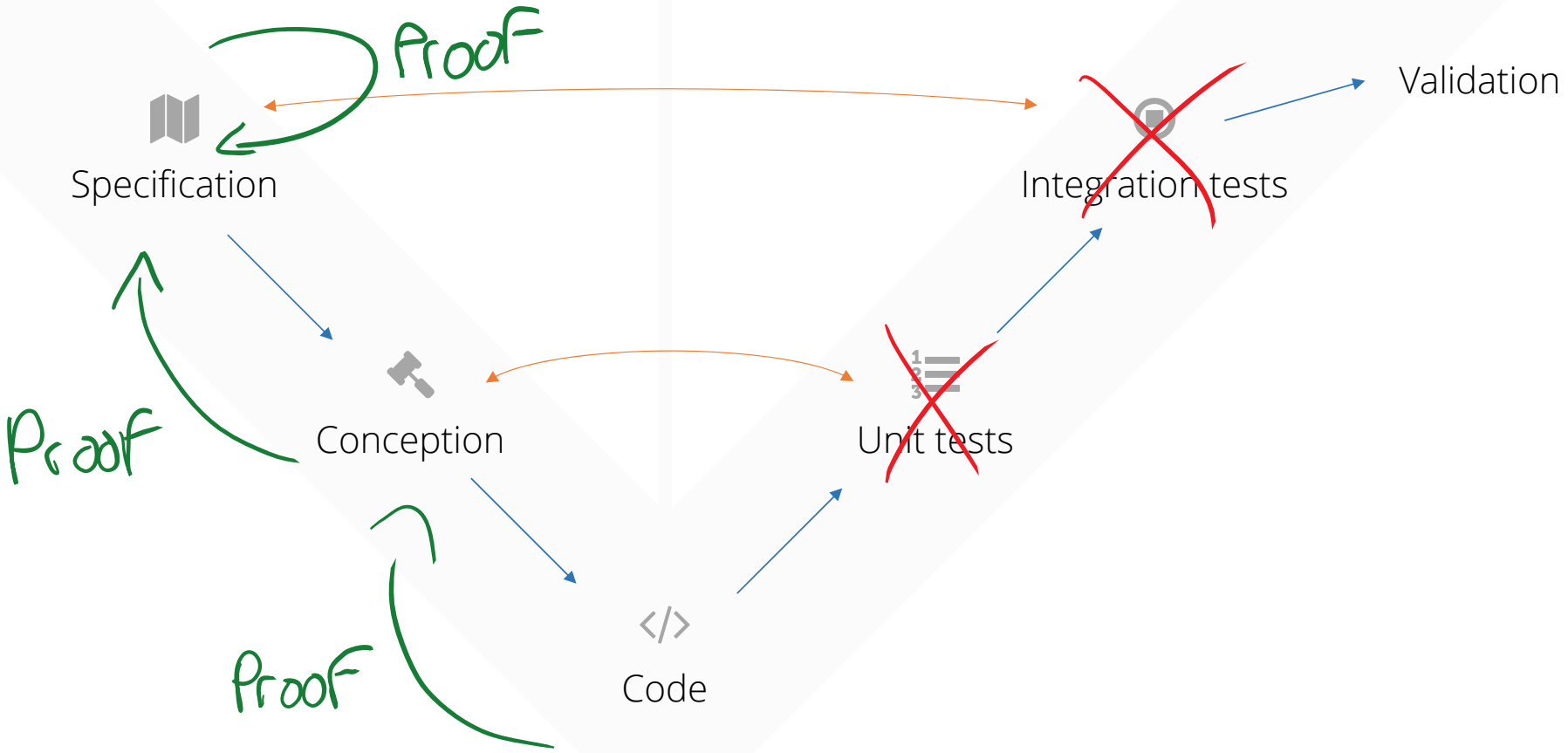
B method development



B method development



B method development



How does method B works

- Module: *modelling of a sub system*
  - Component

How does method B works

- Module: *modelling of a sub system*

- Component

Static part
- *Definition of:* Variables, constants, sets
- List of invariants

Dynamic part
- Initialize variables
- Define operations on variables

Proof
- Static part coherence
- Initializing preserve invariants
- Operations preserve invariants

How does method B works

Module: *modelling of a sub system*

Abstract machine

Component

Static part
- *Definition of:* Variables, constants, sets
- List of invariants

Dynamic part
- Initialize variables
- Define operations on variables

Proof
- Static part coherence
- Initializing preserve invariants
- Operations preserve invariants

How does method B works

Module: *modelling of a sub system*

Abstract machine
Refinements

Component

Static part
Definition of: Variables, constants, sets
List of invariants

Dynamic part
Initialize variables
Define operations on variables

Proof
Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

Module: *modelling of a sub system*

- Abstract machine
- Refinements
- Implementation

→ Component →

Static part

Definition of: Variables, constants, sets
List of invariants

Dynamic part

Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

Module: *modelling of a sub system*

- Abstract machine
- Refinements
- Implementation

Component

We refine a previous component: we make it more precise and specific

Static part

Definition of: Variables, constants, sets
List of invariants

Dynamic part

Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

Machine, refinement, implementation

- Abstract machine
- Refinements
- Implementation

Substitutions

Machine, refinement, implementation

- Abstract machine
- Refinements
- Implementation

Substitutions

Machine, refinement, implementation

Abstract machine

- ✓ Precondition
- ✓ Choice
- ✓ Let bindings
- ✗ Sequencing
- ✓ Become such that
- ✓ Simultaneous operations
- ✗ While loop

Refinements

Implementation

Substitutions

Machine, refinement, implementation

Abstract machine

Predicated-based IF

- Precondition
- Choice
- Let bindings
- Sequencing
- Become such that
- Simultaneous operations
- While loop

Refinements

Implementation

Substitutions

Machine, refinement, implementation

Abstract machine

Predicated-based IF

ANY x WHERE x ∈ S

- Precondition
- Choice
- Let bindings
- Sequencing
- Become such that
- Simultaneous operations
- While loop

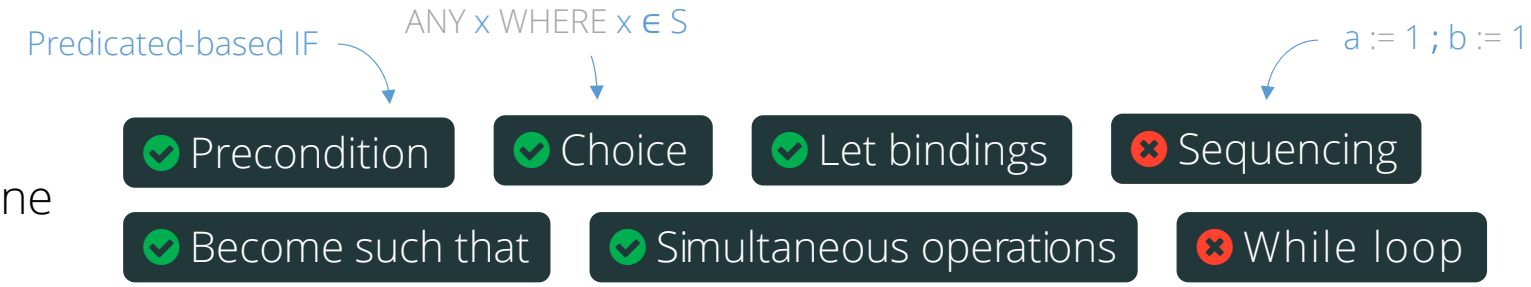
Refinements

Implementation

Substitutions

Machine, refinement, implementation

Abstract machine



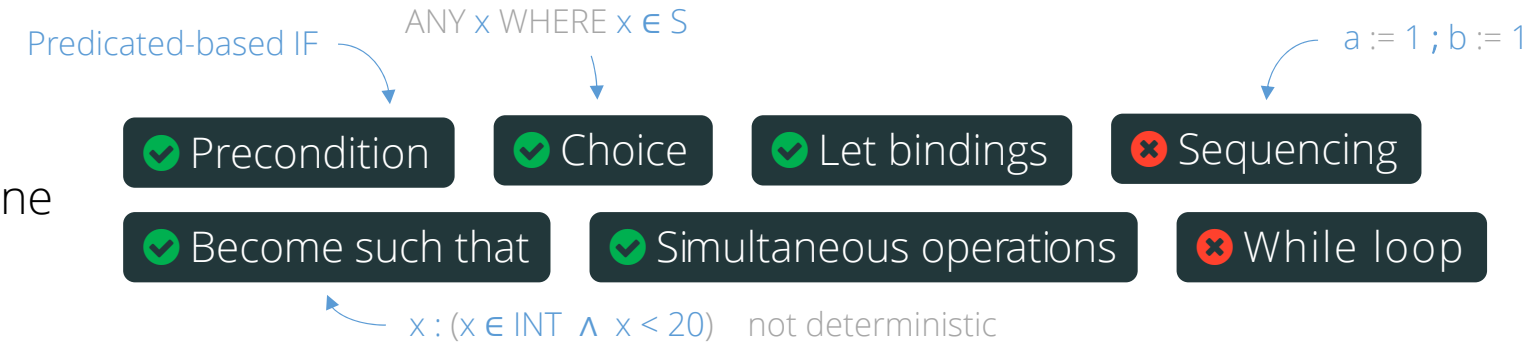
Refinements

Implementation

Substitutions

Machine, refinement, implementation

Abstract machine



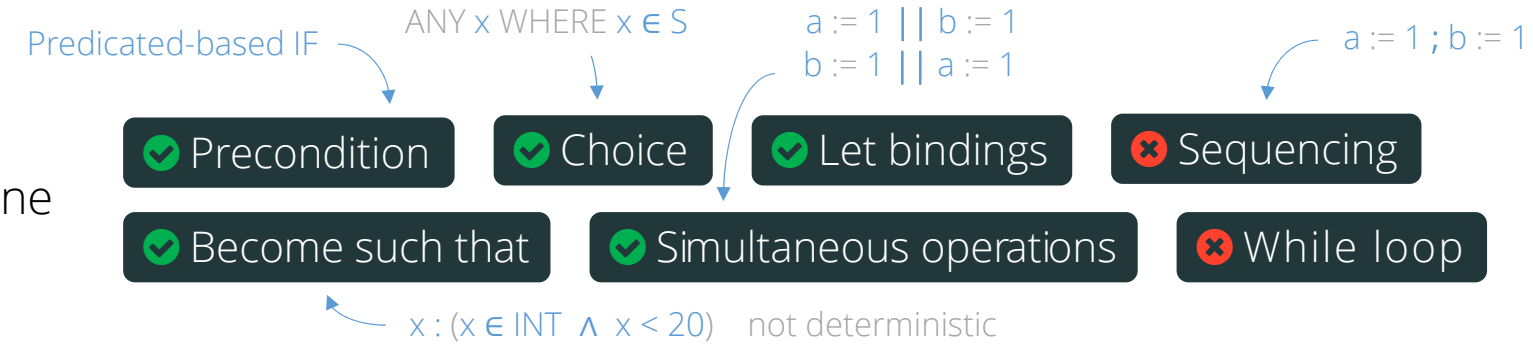
Refinements

Implementation

Substitutions

Machine, refinement, implementation

Abstract machine



Refinements

Implementation

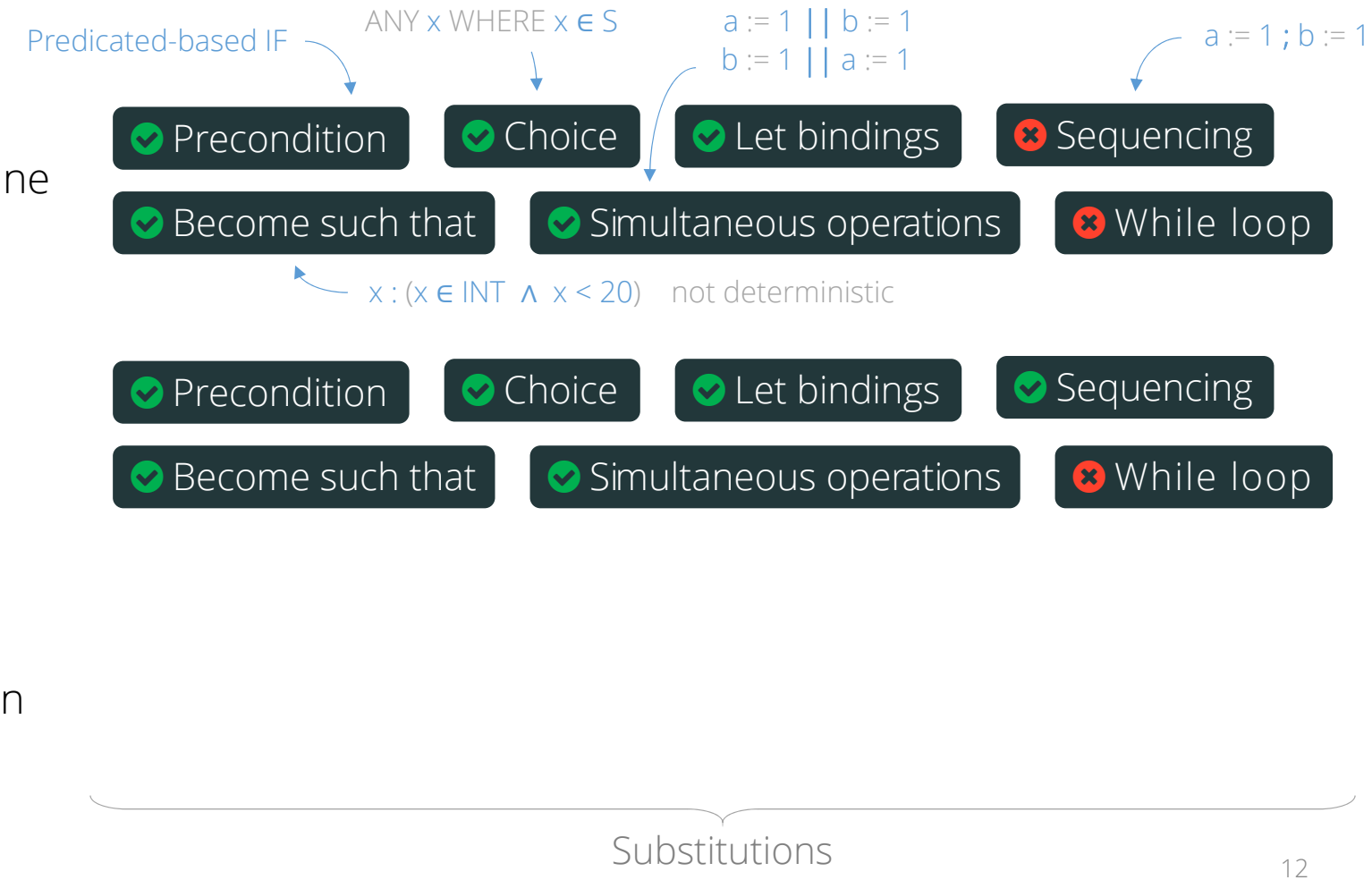
Substitutions

Machine, refinement, implementation

Abstract machine

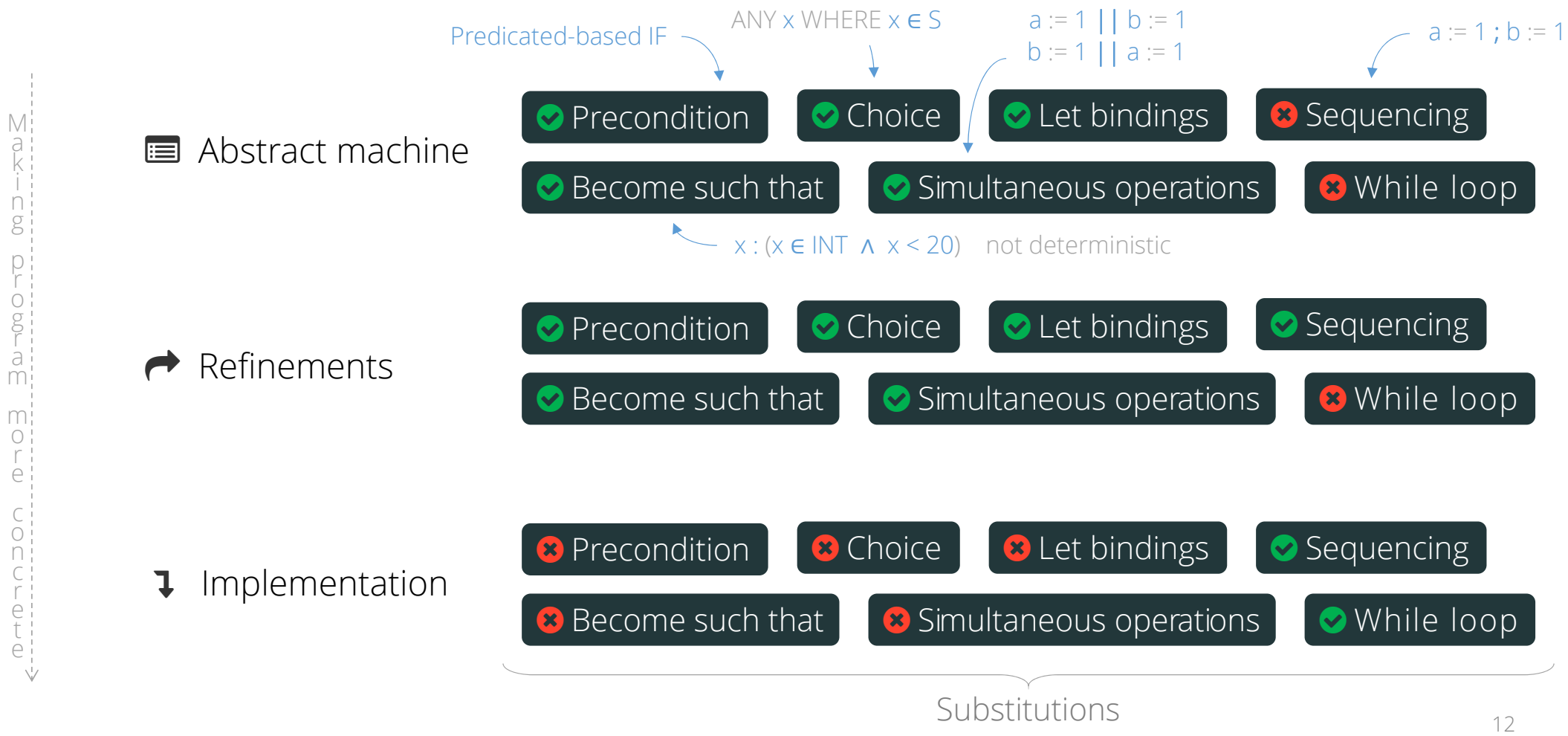
Refinements

Implementation



Machine, refinement, implementation

Machine, refinement, implementation



More substitutions

	Machine	Refinement	Implementation
Block	Y	Y	Y
Identical	Y	Y	Y
Becomes Equal	Y	Y	Y
Precondition	Y	Y	N
Assertion	Y	Y	Y
Bounded choice	Y	Y	N
IF conditional	Y	Y	Y
Conditional Bounded choice	Y	Y	N
Case Conditional	Y	Y	Y
Unbounded choice	Y	Y	N
Local Definition	Y	Y	N
Becomes Element of	Y	Y	N
Becomes such that	Y	Y	N
Local Variable	N	Y	Y
Sequencing	N	Y	Y
Operation Call	Y	Y	Y
While Loop	N	N	Y
Simultaneous	Y	Y	N

B language

B language

State oriented

B language

State oriented

Hoare logic

$$\{P\}\ C\ \{Q\}$$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

$$\frac{}{\{P[E/x]\}\ x \coloneqq E\quad \{P\}}\ \text{ASSIGNMENT}$$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

$$\frac{}{\{P[E/x]\}\ x \coloneqq E\ \ \{P\}}\ \text{ASSIGNMENT}$$

$$\frac{\{P\}\ S\ \{Q\}\,,\ \ \{Q\}\ T\ \{R\}}{\{P\}\ \ S;T\ \ \{R\}}\ \text{COMPOSITION}$$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

$$\frac{}{\{P[E/x]\}\ x \coloneqq E\ \ \{P\}}\ \text{ASSIGNMENT}$$

$$\frac{\{P\}\ S\ \{Q\}\,,\ \ \{Q\}\ T\ \{R\}}{\{P\}\ \ S;T\ \ \{R\}}\ \text{COMPOSITION}$$

$$\frac{\{B \wedge P\}\ S\ \{Q\}\,,\ \ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{end}\ \ \{Q\}}\ \text{CONDITIONAL}$$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

$$\frac{}{\{P[E/x]\}\ x \coloneqq E\ \ \{P\}}\ \text{ASSIGNMENT}$$

$$\frac{\{P\}\ S\ \{Q\}\,,\ \ \{Q\}\ T\ \{R\}}{\{P\}\ \ S;T\ \ \{R\}}\ \text{COMPOSITION}$$

$$\frac{\{B \wedge P\}\ S\ \{Q\}\,,\ \ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{end}\ \ \{Q\}}\ \text{CONDITIONAL}$$

$$\frac{\{I \wedge B\}\ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \ \{\neg B \wedge I\}}\ \text{WHILE}$$

B language

State oriented

Hoare logic

$\{P\}\ C\ \{Q\}$

$\{true\}$ y := 3 $\{y = 3\}$

$\{x > 2\}$ x := x*x $\{x > 4\}$

$$\frac{}{\{P[E/x]\}\ x \coloneqq E\ \ \{P\}}\ \text{ASSIGNMENT}$$

$$\frac{\{P\}\ S\ \{Q\},\ \ \{Q\}\ T\ \{R\}}{\{P\}\ \ S;T\ \ \{R\}}\ \text{COMPOSITION}$$

$$\frac{\{B \wedge P\}\ S\ \{Q\},\ \ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{end}\ \ \{Q\}}\ \text{CONDITIONAL}$$

$$\frac{P\ \rightarrow P',\ \ \{P'\}\ S\ \{Q'\},\ \ Q' \rightarrow Q}{\{P\}\ \ S\ \ \{Q\}}\ \text{CONSEQUENCE}$$

$$\frac{\{I \wedge B\}\ S\ \ \{I\}}{\{I\}\ \ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \ \{\neg B \wedge I\}}\ \text{WHILE}$$

B language

B language

Arithmetic

B language

Arithmetic $+ \ - \ \div \ x^y \ \times \ < \ > \ \leq \ \geq$

B language

Arithmetic $+ \quad - \quad \div \quad x^y \quad \times \quad < \quad > \quad \leq \quad \geq \qquad \mathbb{Z} \quad \mathbb{N}$

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq$ $\mathbb{Z} \; \mathbb{N}$ $\prod \; \sum$

B language

Arithmetic $+ \ - \ \div \ x^y \ \times \ < \ > \ \leq \ \geq$ $\mathbb{Z} \ \mathbb{N}$ $\prod \ \sum$

Functions and relations

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

Set comprehension, generalized union & intersections

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

Set comprehension, generalized union & intersections

Records

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq \qquad \mathbb{Z} \; \mathbb{N} \qquad \prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

Set comprehension, generalized union & intersections

Records

Trees

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq$ $\mathbb{Z} \; \mathbb{N}$ $\prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

Set comprehension, generalized union & intersections

Records

Trees

Sequences

B language

Arithmetic $+ \; - \; \div \; x^y \; \times \; < \; > \; \leq \; \geq$ $\mathbb{Z} \; \mathbb{N}$ $\prod \; \sum$

Functions and relations

Partial / total functions, surjections, lambda, domain/range manipulations, closure, inversions…

Sets

Set comprehension, generalized union & intersections

Records

Trees

Sequences

No algebraic data types

Proofs with B

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

How does B method handles proofs?

Component

- Abstract machine
- Refinements
- Implementation

```
MACHINE
    Name(input1, input2, ...)
```

How does B method handles proofs?

Component

- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
```

How does B method handles proofs?

Component

- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
```

How does B method handles proofs?

Component

- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
```

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
```

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
```

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
```

lemmas

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
```

Prove all predicates

lemmas

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
INITIALISATION
   var1 := expr || var2 := expr
```

Prove all predicates

lemmas

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
INITIALISATION
   var1 := expr || var2 := expr
```

Prove all predicates

For each initialization, prove invariant conservations

lemmas

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
INITIALISATION
   var1 := expr || var2 := expr
OPERATIONS
   varOutput1 ← fun1(i1, i2, ...) =
      ...
   varOutput2 ← fun2(i1, i2, ...) =
      ...
```

Prove all predicates

For each initialization, prove invariant conservations

lemmas

How does B method handles proofs?

Component
- Abstract machine
- Refinements
- Implementation

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
INITIALISATION
   var1 := expr || var2 := expr
OPERATIONS
   varOutput1 ← fun1(i1, i2, ...) =
      ...
   varOutput2 ← fun2(i1, i2, ...) =
      ...
```

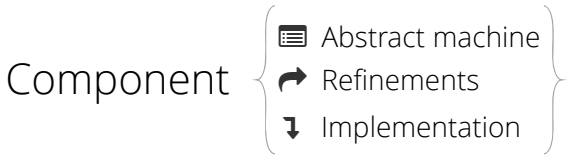
Prove all predicates

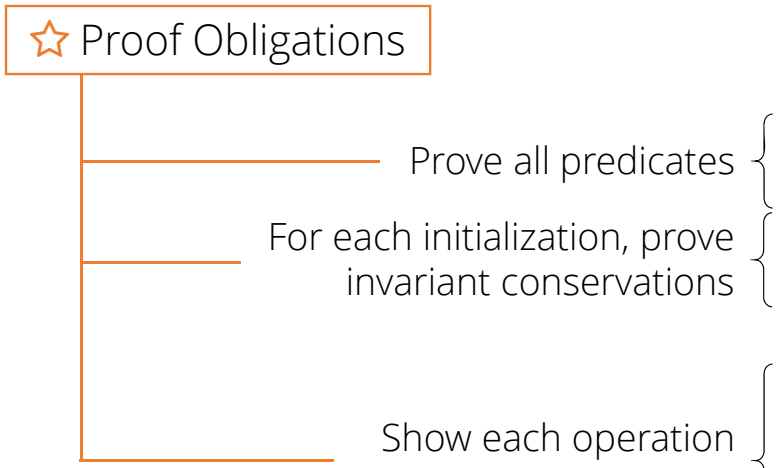
For each initialization, prove invariant conservations

Show each operation conserve invariants

lemmas

How does B method handles proofs?

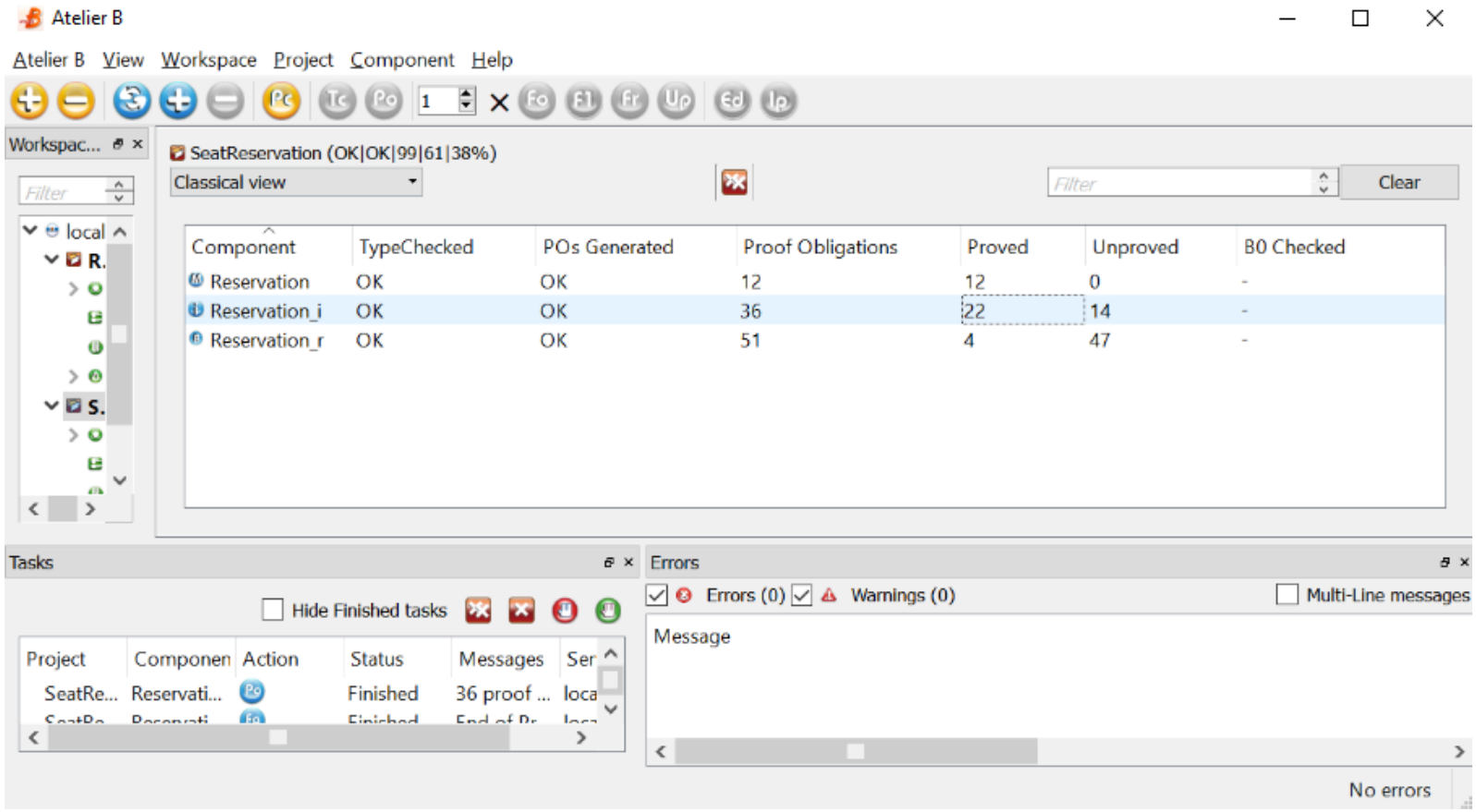




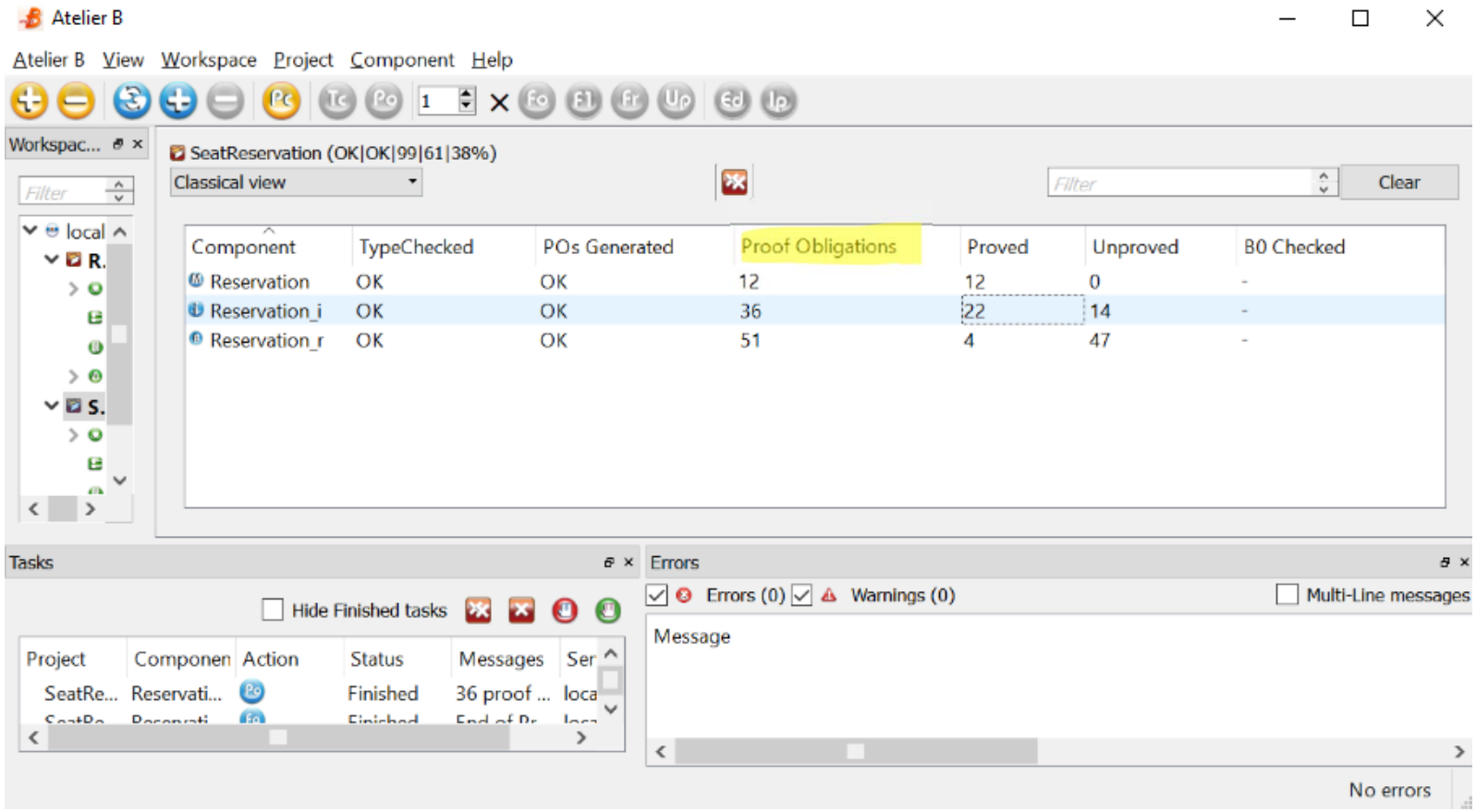

```
MACHINE
   Name(input1, input2, ...)
CONSTRAINTS
   input1 ∈ INT ∧ input2 ∈ INT ...
CONSTANTS
   cst1, cst2, ...
VARIABLES
   var1, var2, ...
INVARIANT
   var1 + var2 ∈ {x . X**(1/2) ∈ ℤ} ∧ ...
ASSERTIONS
   predicate1 ∧ predicate2 ∧ ...
INITIALISATION
   var1 := expr || var2 := expr
OPERATIONS
   varOutput1 ← fun1(i1, i2, ...) =
      ...
   varOutput2 ← fun2(i1, i2, ...) =
      ...
```

lemmas

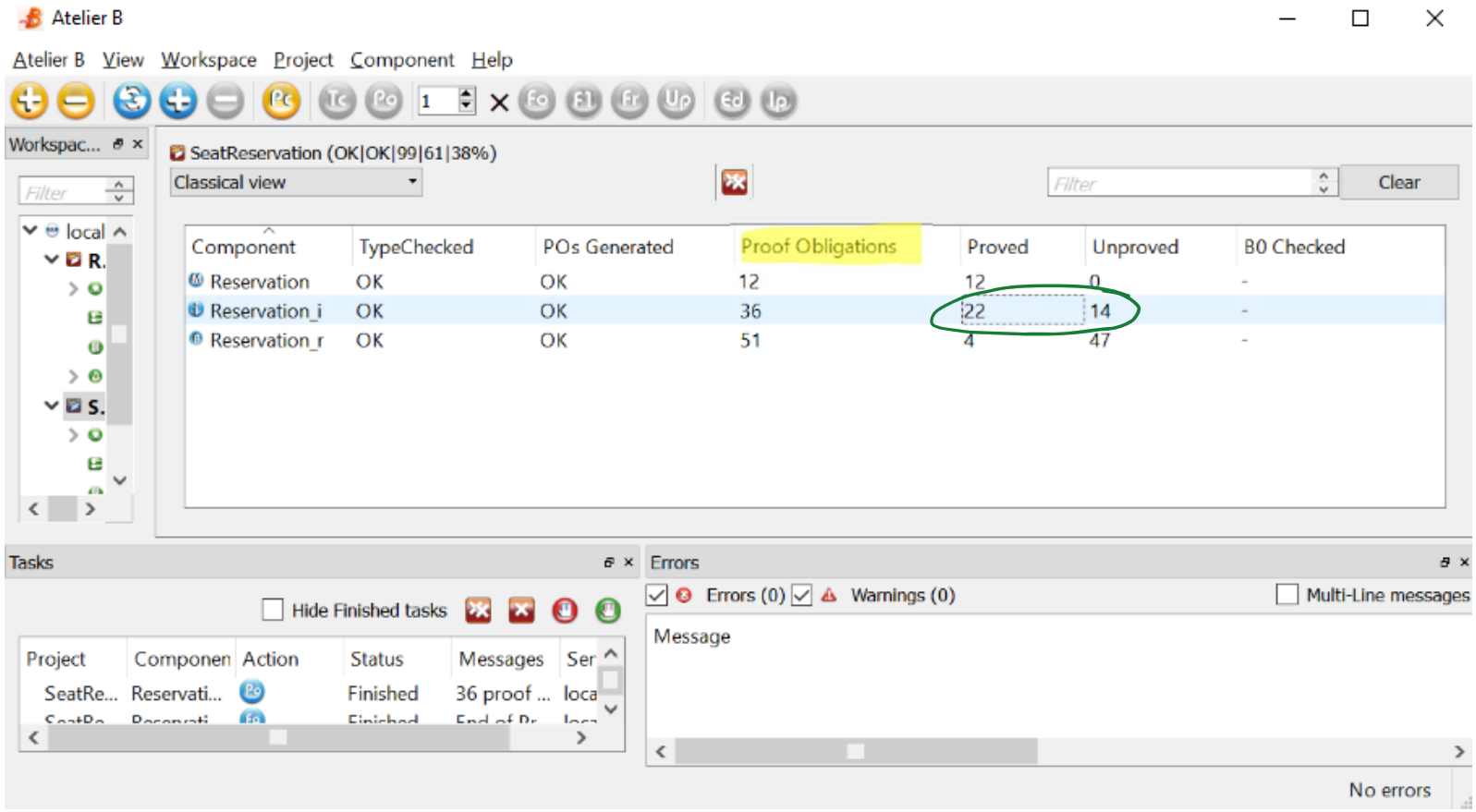
Proving with B method: interactive only



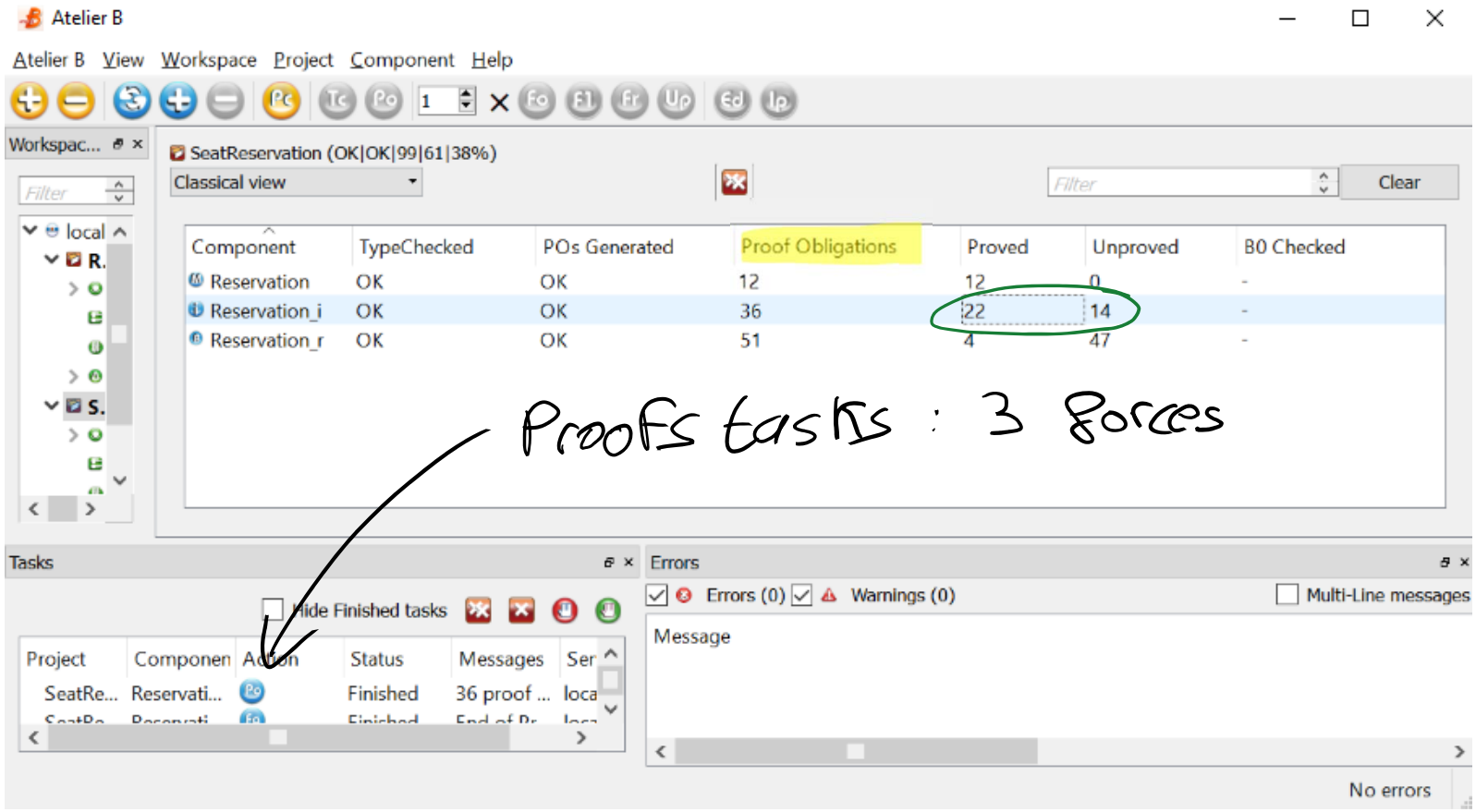
Proving with B method: interactive only



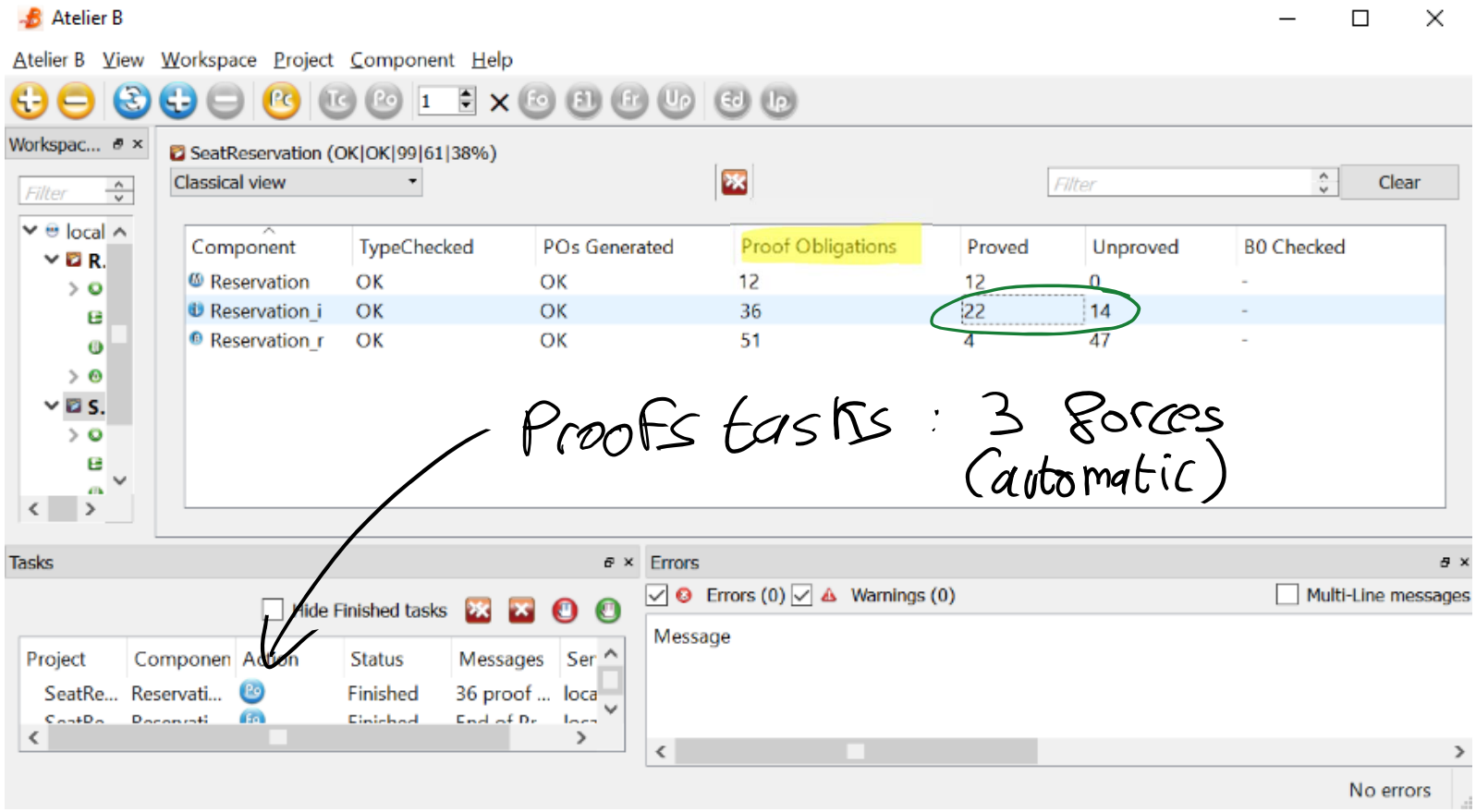
Proving with B method: interactive only



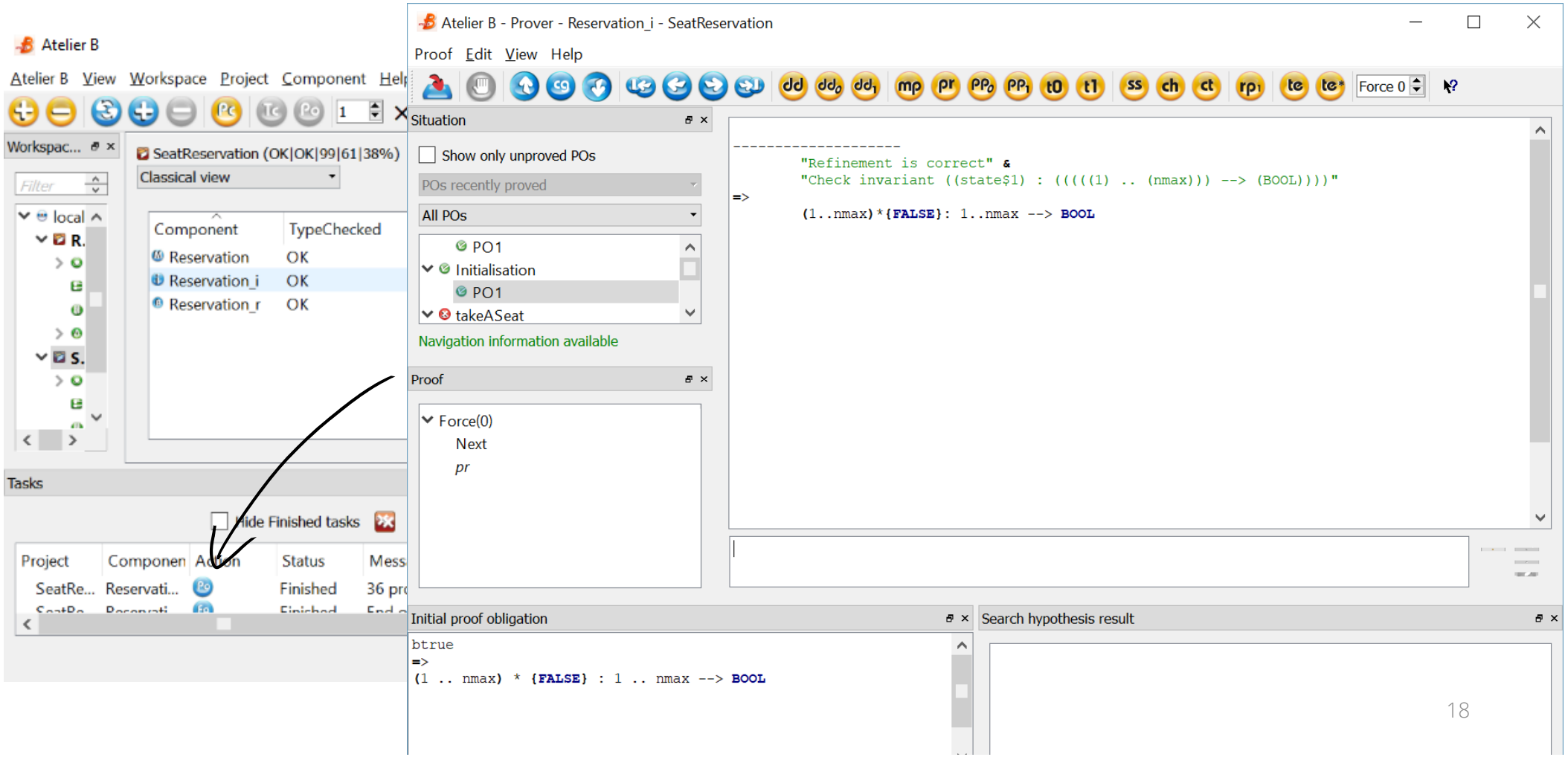
Proving with B method: interactive only



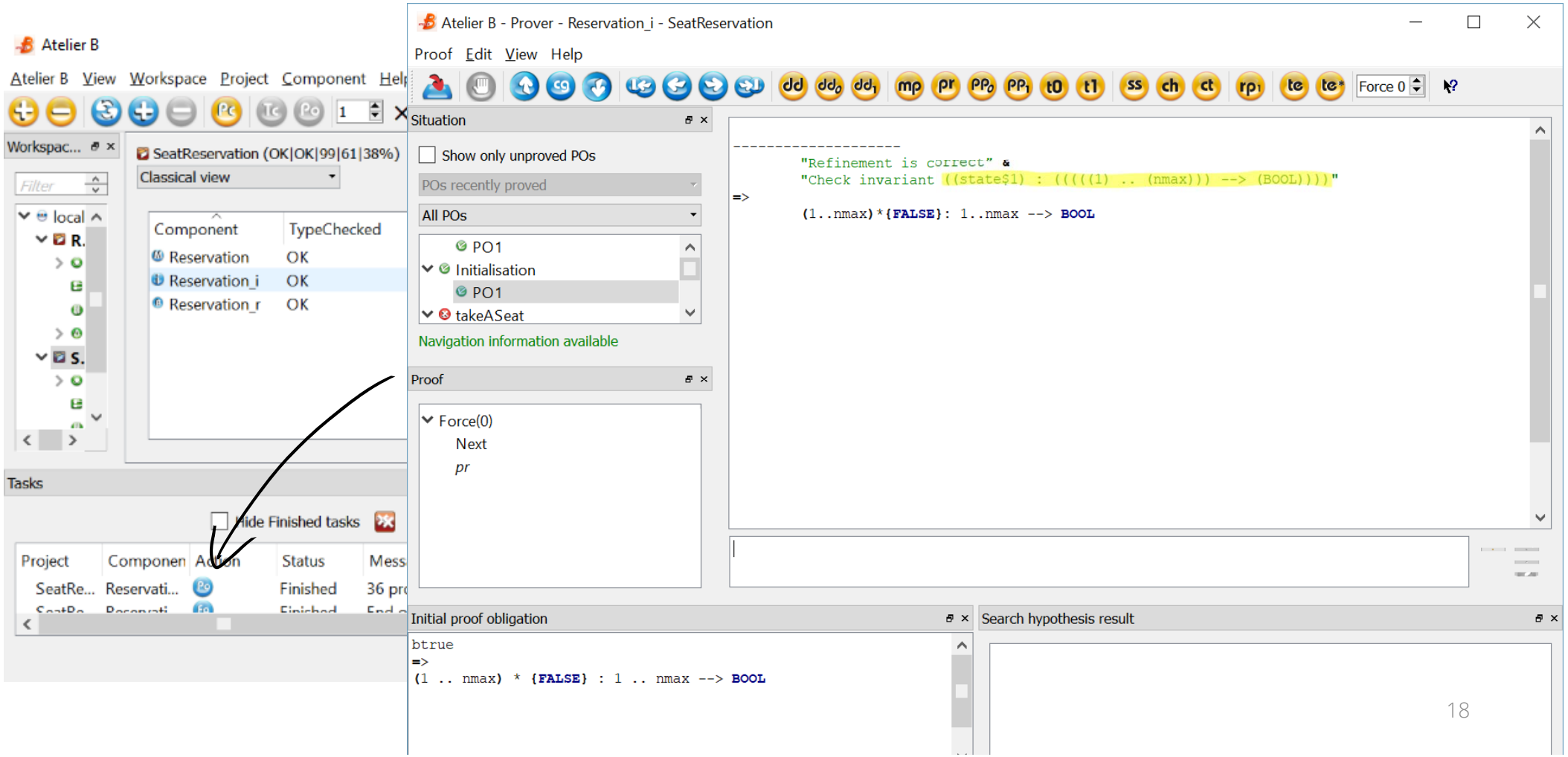
Proving with B method: interactive only



Proving with B method: interactive only



Proving with B method: interactive only



How does B method internally prove things?

First, expressions are normalized

Expression	Normal Form
$n > m$	$m + 1 <= n$
$m < n$	$m + 1 <= n$
$a \ <=> \ b$	$(a => b)\&(b => a)$
$a \ <: \ b$	$a : POW(b)$
$a \ <<: \ b$	$a : POW(b)\&not(a = b)$
$a \ /: \ b$	$not(a : b)$
$a \ /= \ b$	$not(a = b)$
$a \ /<: \ b$	$not(a : POW(b))$
$a \ /<<: \ b$	$a : POW(b) => a = b$
$a : NATURAL$	$a : INTEGER\&0 <= a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E)/\backslash(NATURAL - \{0\} + - >> E)$
$<>$	$\{\}$
$\{x, y\}$	$\{x\}\backslash/\{y\}$
$\{x \mid P\}$	$SET(x).P$

How does B method internally prove things?

First, expressions are normalized

$a > b \longrightarrow b + 1 \leq a$

Expression	Normal Form
$n > m$	$m + 1 <= n$
$m < n$	$m + 1 <= n$
$a <=> b$	$(a => b)\&(b => a)$
$a <: b$	$a : POW(b)$
$a <<: b$	$a : POW(b)\&not(a = b)$
$a /: b$	$not(a : b)$
$a /= b$	$not(a = b)$
$a /<: b$	$not(a : POW(b))$
$a /<<: b$	$a : POW(b) => a = b$
$a : NATURAL$	$a : INTEGER\&0 <= a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E)/\backslash(NATURAL - \{0\} + - >> E)$
$<>$	$\{\}$
$\{x, y\}$	$\{x\}\backslash/\{y\}$
$\{x \mid P\}$	$SET(x).P$

How does B method internally prove things?

First, expressions are normalized

$a > b \longrightarrow b + 1 \leq a$

$S \subseteq \{1,2,3\} \longrightarrow S \in POW(\{1\} \cup \{2\} \cup \{3\})$

Expression	Normal Form
$n > m$	$m + 1 <= n$
$m < n$	$m + 1 <= n$
$a <=> b$	$(a => b)\&(b => a)$
$a <: b$	$a : POW(b)$
$a <<: b$	$a : POW(b)\&not(a = b)$
$a /: b$	$not(a : b)$
$a /= b$	$not(a = b)$
$a /<: b$	$not(a : POW(b))$
$a /<<: b$	$a : POW(b) => a = b$
$a : NATURAL$	$a : INTEGER\&0 <= a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E)/\backslash(NATURAL - \{0\} + - >> E)$
$<>$	$\{\}$
$\{x, y\}$	$\{x\}\backslash/\{y\}$
$\{x \mid P\}$	$SET(x).P$

How does B method internally prove things?

Theory files: list of rules

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
       =>
       #x.(a & b) == (#x.a & b)
```

How does B method internally prove things?

Theory files: list of rules

10.000 lines of rules

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
       =>
       #x.(a & b) == (#x.a & b)
```

How does B method internally prove things?

Theory files: list of rules

10.000 lines of rules

User can add rules

⚠ No safe core

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
       =>
       #x.(a & b) == (#x.a & b)
```

How does B method internally prove things?

Theory files: list of rules

10.000 lines of rules

User can add rules

⚠ No safe core

Still, we can prove a rule before adding it

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
       =>
       #x.(a & b) == (#x.a & b)
```

How does B method internally prove things?

Theory files: list of rules

10.000 lines of rules

User can add rules

⚠ No safe core

Still, we can prove a rule before adding it

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       P == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
```

How does B method internally prove things?

Theory files: list of rules

10.000 lines of rules

User can add rules

⚠ No safe core

Still, we can prove a rule before adding it

Automatic prover

- Applies recursively rules
- Case proof
- Tactics

```
THEORY SimplifyX IS
       (i => (j => k)) == (i & j => k)
;
       (a => bfalse) == not(a)
;
       (bfalse => a) == btrue
;
       (a => btrue) == btrue
;
       P == btrue
;
       (btrue => a) == a
;
       (bvrb(x)) &
       (x\a)
       =>
       #x.a == a
;
       (bvrb(x)) &
       (x\b)
```

Interactive proofs

- Proof construction
- Search information
- Browsing proof obligations
- Command repetition

Interactive proofs

Proof construction

Search information

Browsing proof obligations

Command repetition Repeat `(rr)`, loop `(bb)`...

Interactive proofs

Proof construction

Search information

Browsing proof obligations — Back `(ba)`, Reset `(re)`, Next `(ne)`, Previous `(pv)`, Goto `(go)`…

Command repetition — Repeat `(rr)`, loop `(bb)`…

Interactive proofs

Proof construction	
Search information	Search rule / hypothesis / goal, show proof, reduce PO…
Browsing proof obligations	Back `(ba)`, Reset `(re)`, Next `(ne)`, Previous `(pv)`, Goto `(go)`…
Command repetition	Repeat `(rr)`, loop `(bb)`…

Interactive proofs

Proof construction

Interactive proofs

Proof construction

Prover call *automatic proofs & simplification*

Interactive proofs

Proof construction

Prover call *automatic proofs & simplification* **pr**

Interactive proofs

Proof construction

Prover call *automatic proofs & simplification* `pr` `ss`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr` `ss`
Rule applications		

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr` `ss`
Rule applications		`ar`

Interactive proofs

Proof construction

Prover call *automatic proofs & simplification*	`pr`	`ss`
Rule applications	`ar`	
Rewrite *equality applications, abstract terms*		

Interactive proofs

Proof construction

Prover call *automatic proofs & simplification*	`pr`	`ss`
Rule applications	`ar`	
Rewrite *equality applications, abstract terms*	`eh`	

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`

$$a + 3 < b \ \wedge \ a + 3 \in S$$
$$\Downarrow$$
$$n < b \ \wedge n \in S$$

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`
Inference rules	*contradiction, false hypothesis, cases, instantiate*		

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`
Rule applications		`ar`	
Rewrite	*equality applications, abstract terms*	`eh`	`ae`
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`	
Rule applications		`ar`		
Rewrite	*equality applications, abstract terms*	`eh`	`ae`	
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`	`dc`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`		
Rule applications		`ar`			
Rewrite	*equality applications, abstract terms*	`eh`	`ae`		
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`	`dc`	`se`

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`		
Rule applications		`ar`			
Rewrite	*equality applications, abstract terms*	`eh`	`ae`		
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`	`dc`	`se`
Operations on hypothesis	*deduction, add hypothesis…*				

Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`		
Rule applications		`ar`			
Rewrite	*equality applications, abstract terms*	`eh`	`ae`		
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`	`dc`	`se`
Operations on hypothesis	*deduction, add hypothesis…*	`dd`			

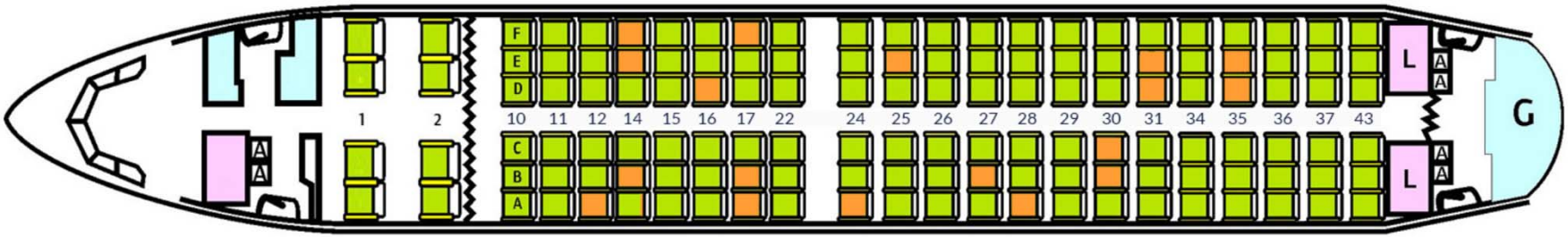
Interactive proofs

Proof construction

Prover call	*automatic proofs & simplification*	`pr`	`ss`		
Rule applications		`ar`			
Rewrite	*equality applications, abstract terms*	`eh`	`ae`		
Inference rules	*contradiction, false hypothesis, cases, instantiate*	`ct`	`fh`	`dc`	`se`
Operations on hypothesis	*deduction, add hypothesis…*	`dd`	`ah`		

Concrete case

Example: seat reservation system



Two operations: *reserve a seat* or *free a seat*

Data : *set of seats* and *sub set of taken seats*

Thank you for listening!

Any questions?