

B Method

Proof assistants

May 16, 2017

Lucas Franceschino

What is B method?

© B-method goal



Specifications

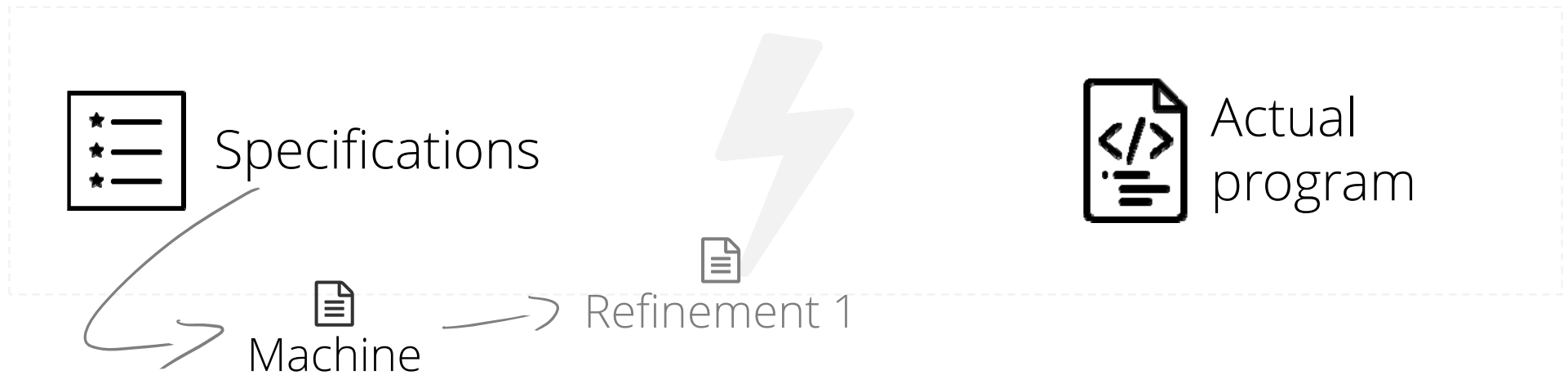


Actual
program

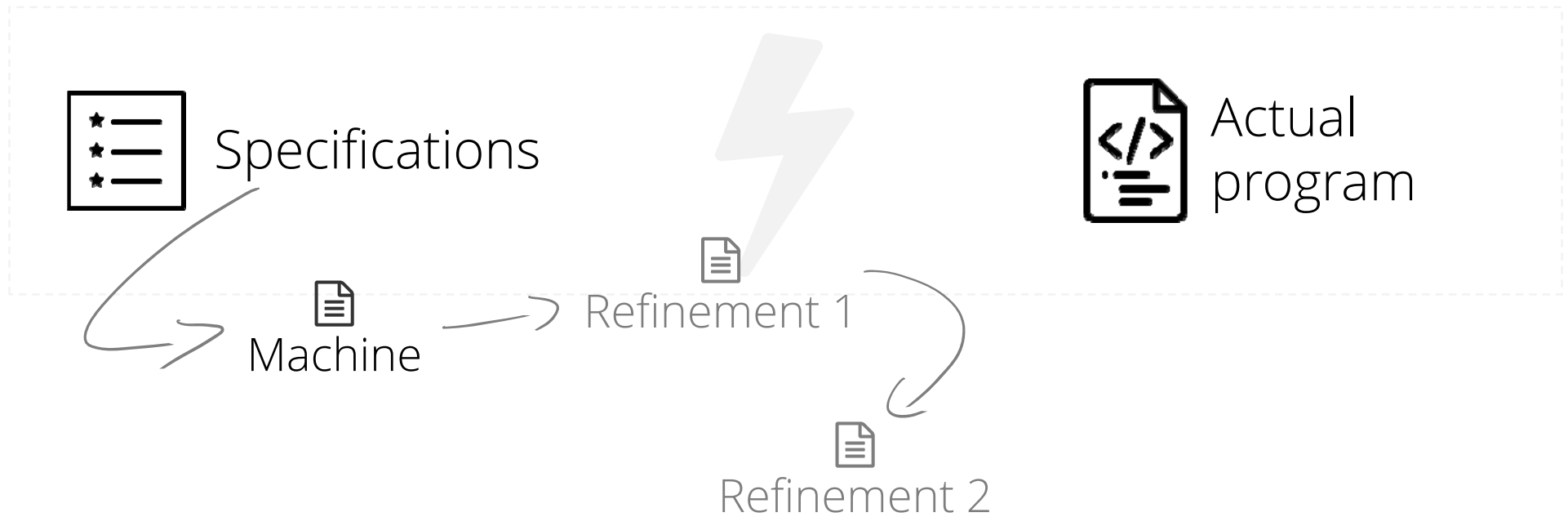
© B-method goal



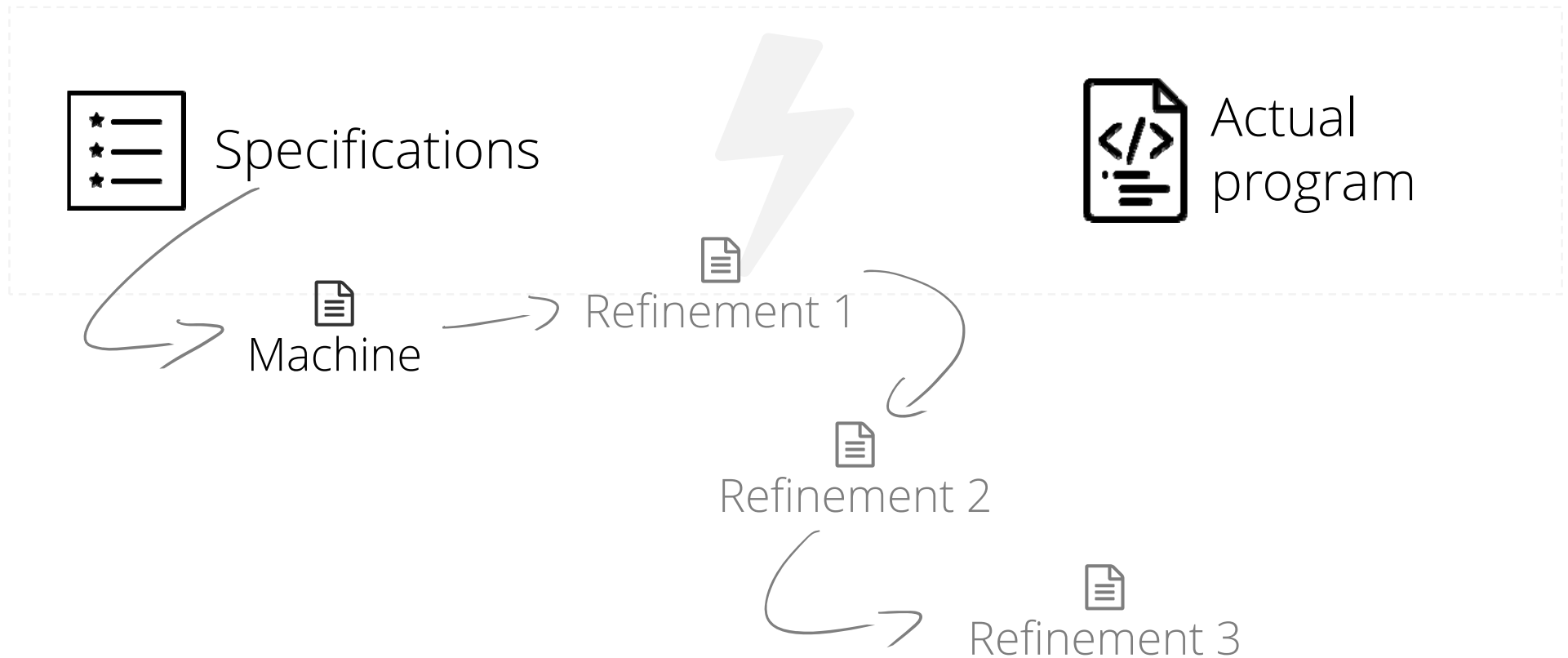
© B-method goal



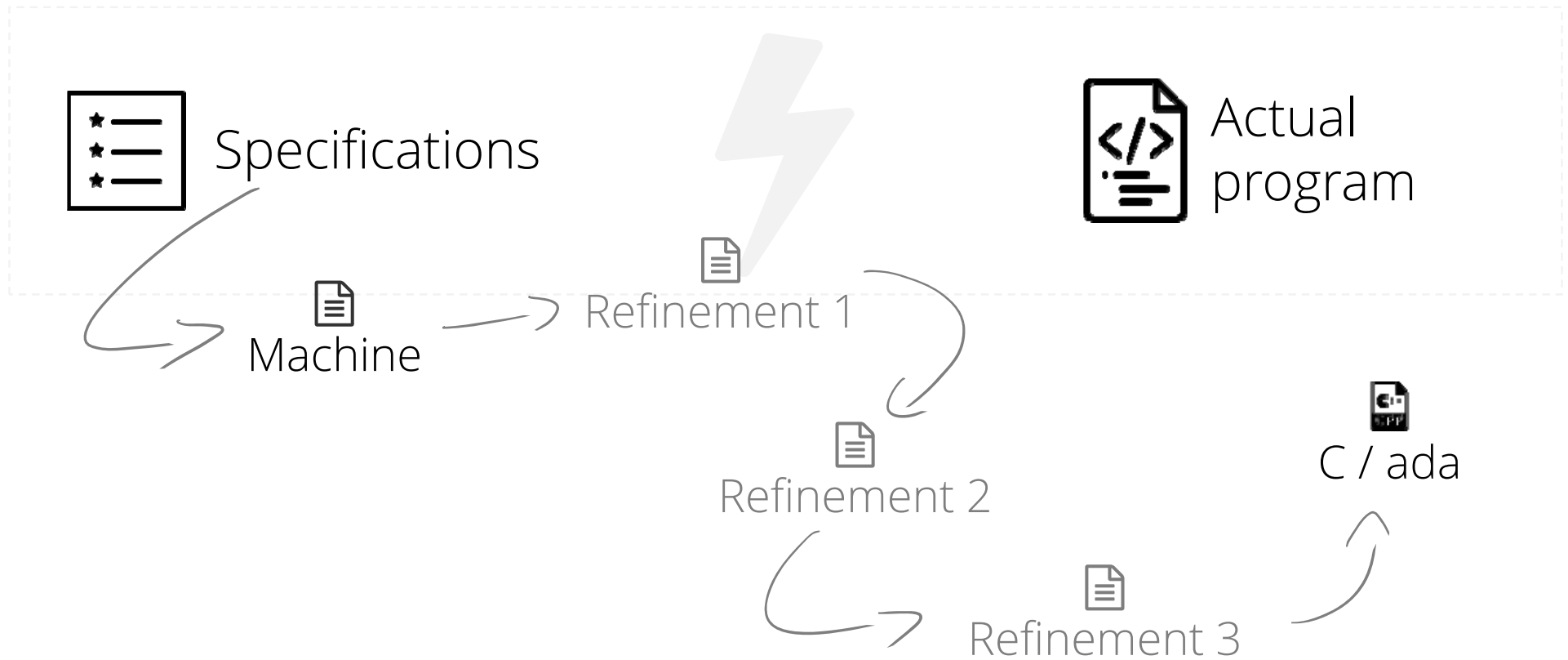
© B-method goal



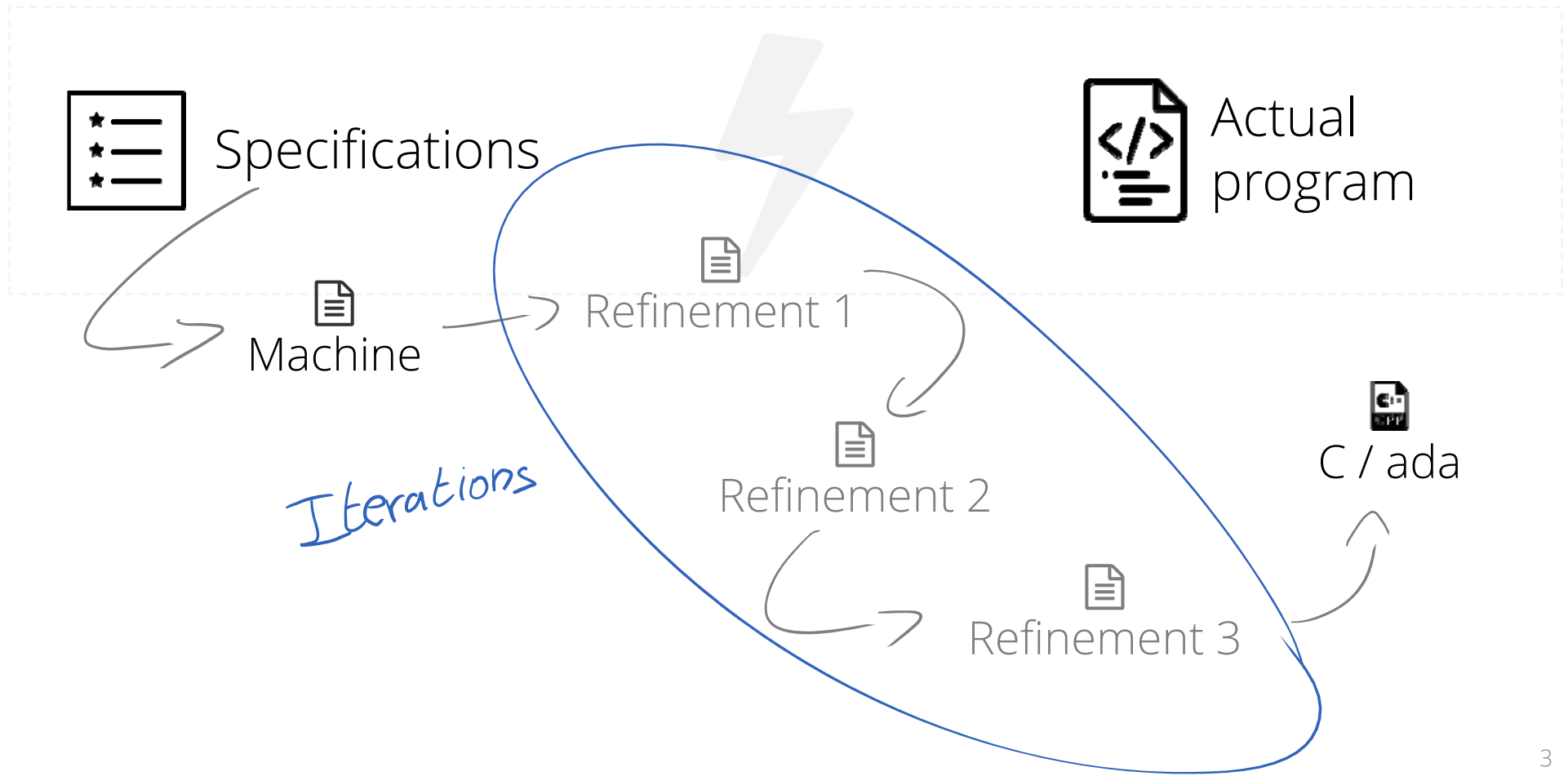
© B-method goal



© 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"

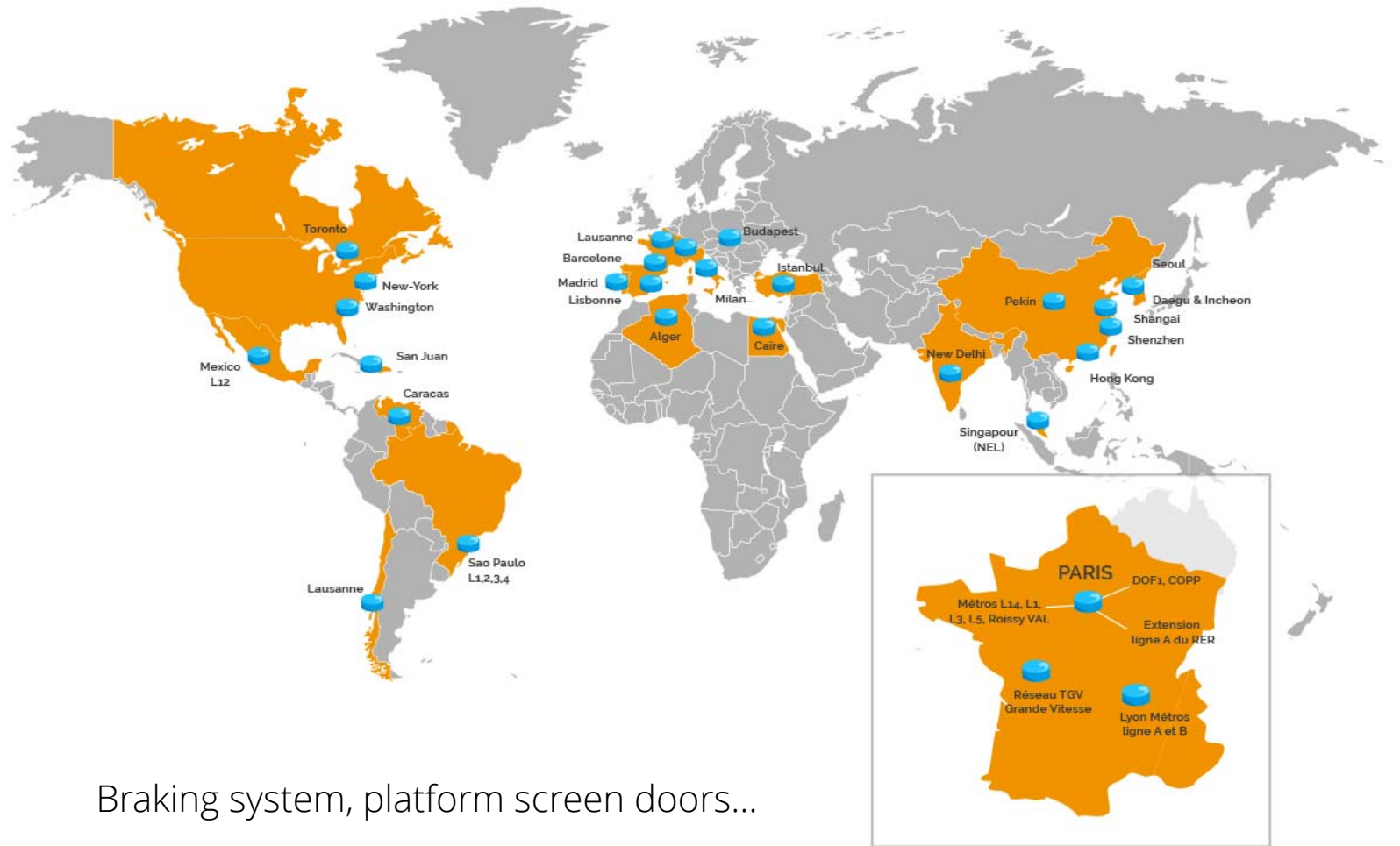
Rodin platform

↳ Was in the development team of Ada

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

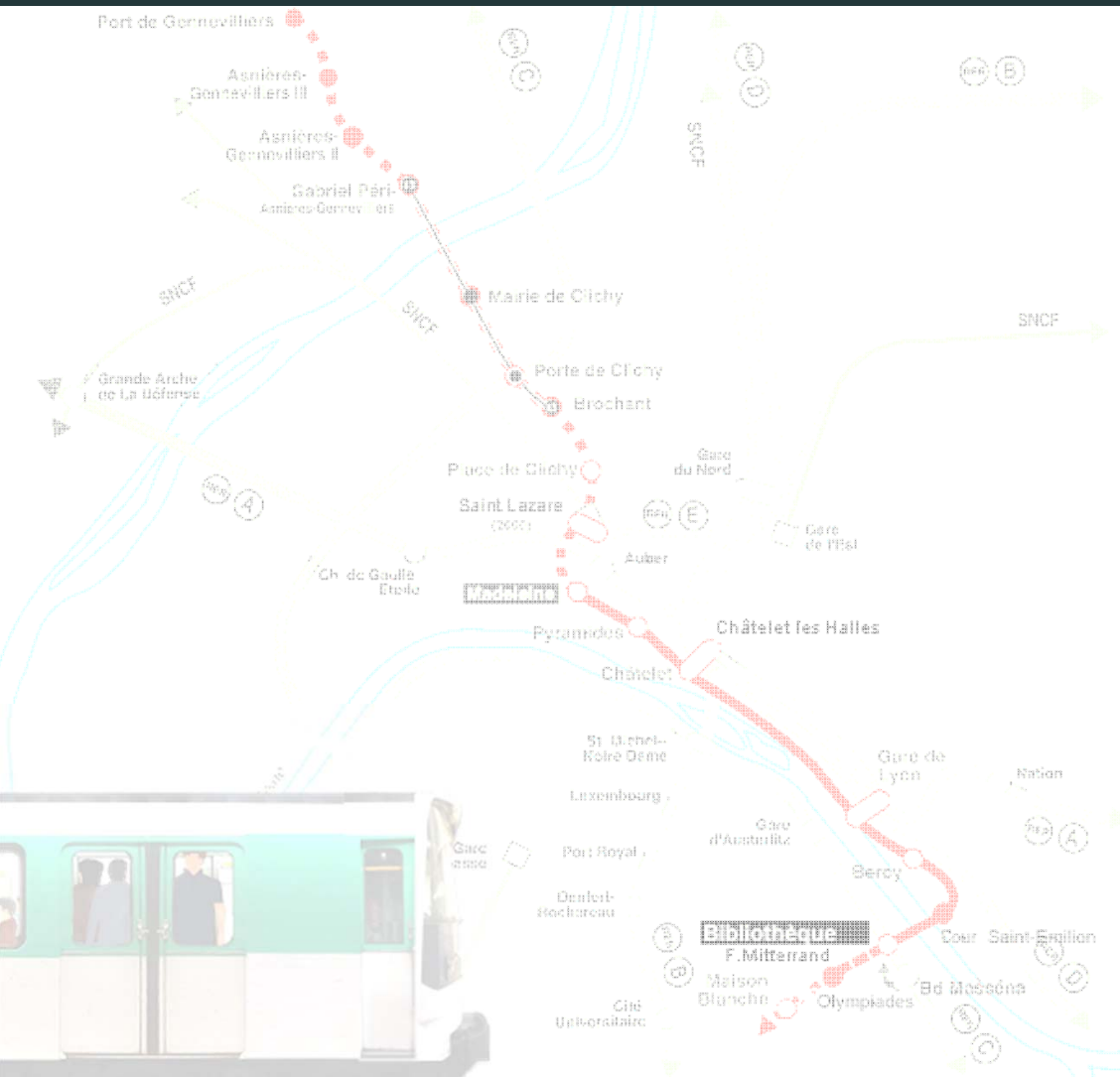
Use cases

Train related B projects around the world



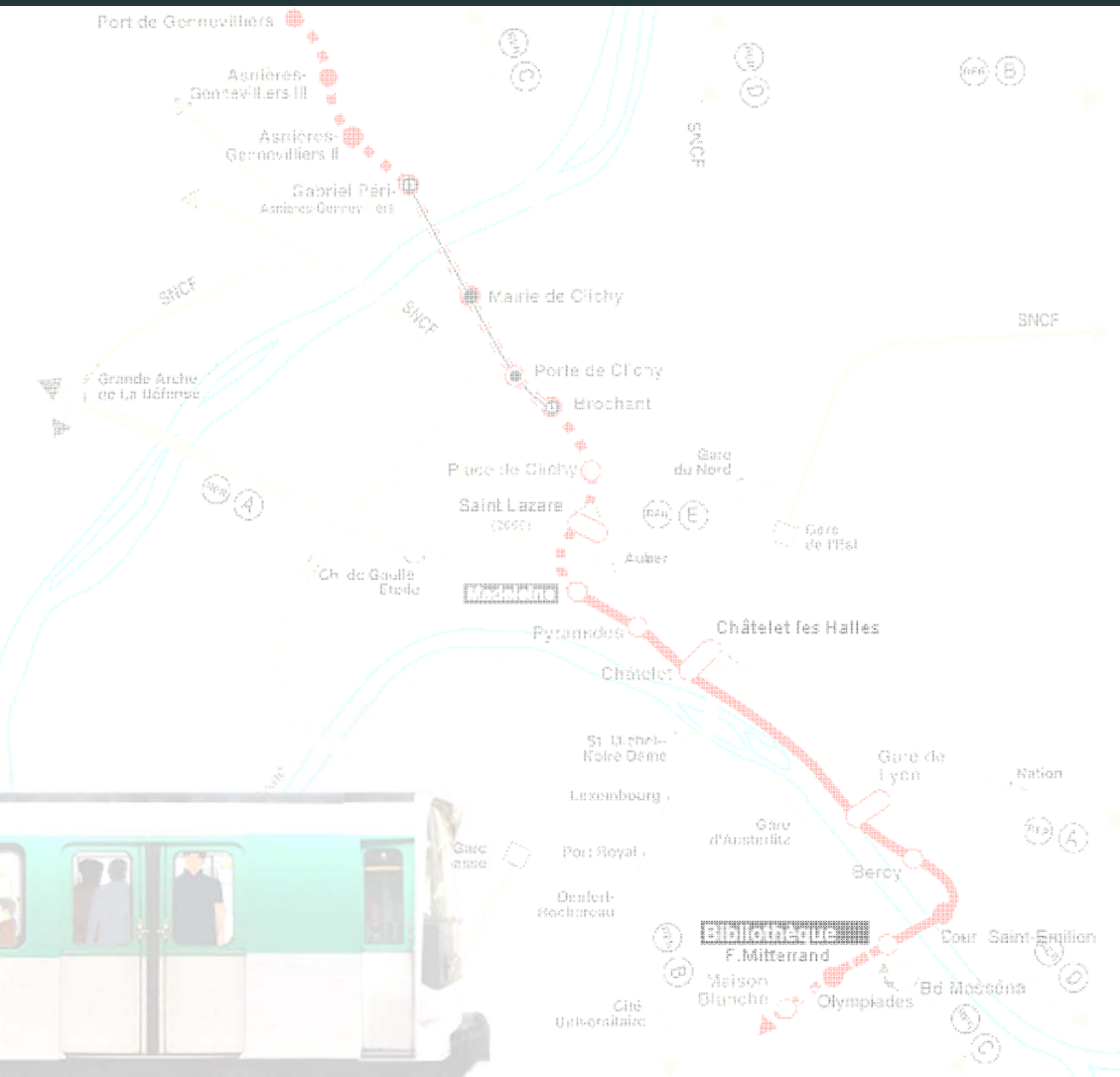
Braking system, platform screen doors...

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 👍

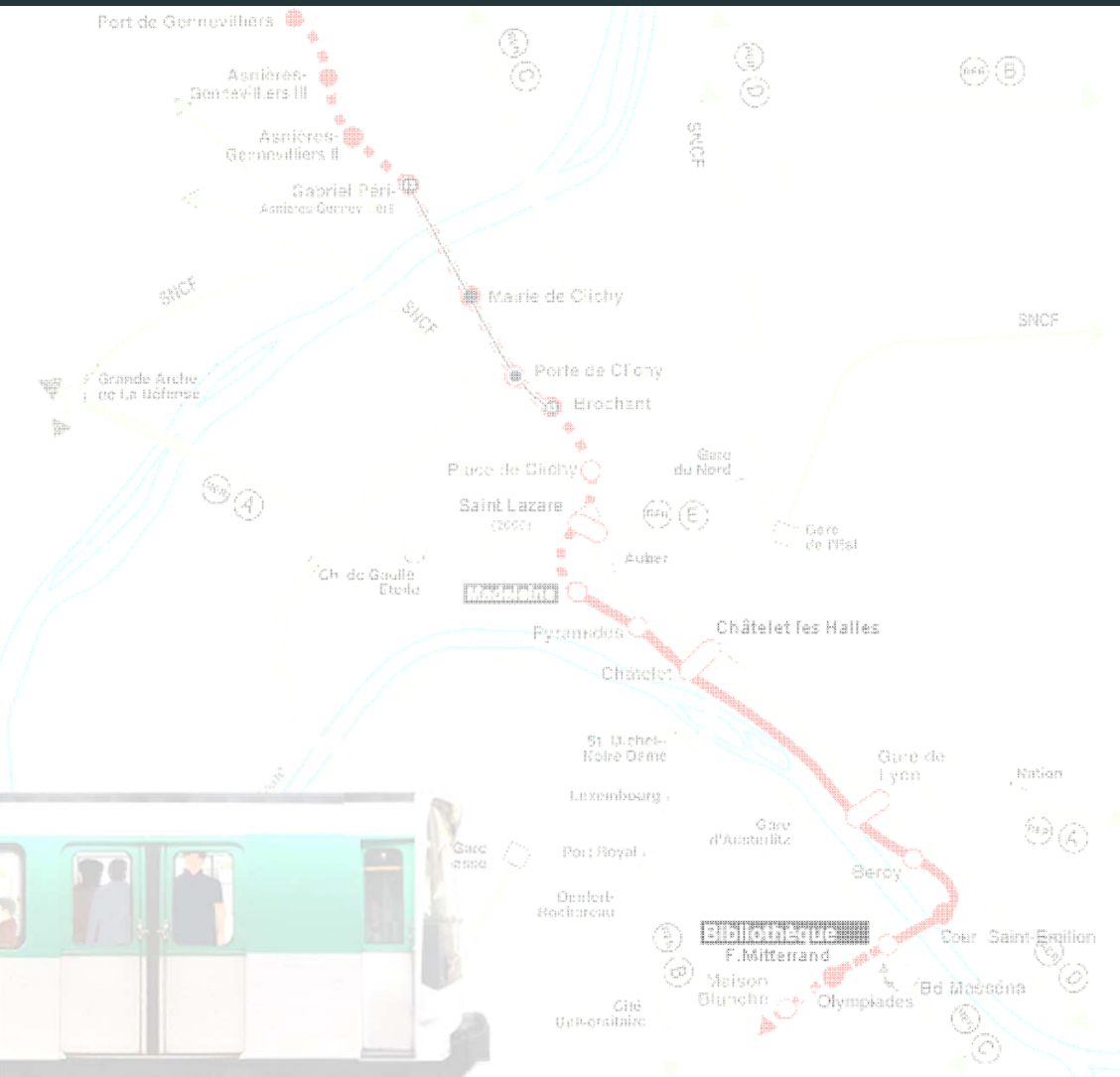
87 000 lines of Ada </>

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 </>

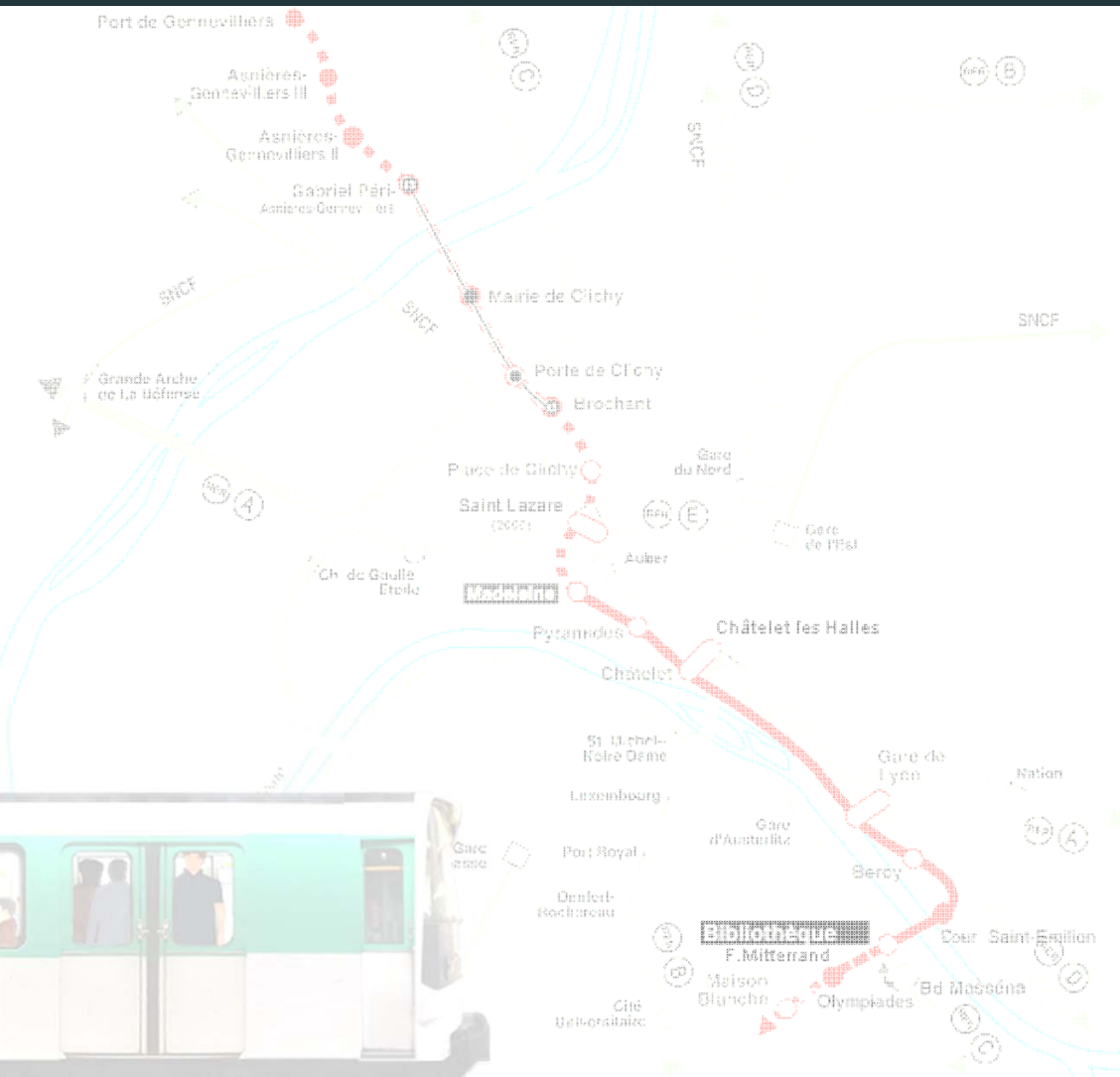
9.2 km 🚂

October 1998 📅

Driverless trains 🔄

Extension in 2003 ↗️

80 million passengers in 2009 🧑🧑🧑



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 

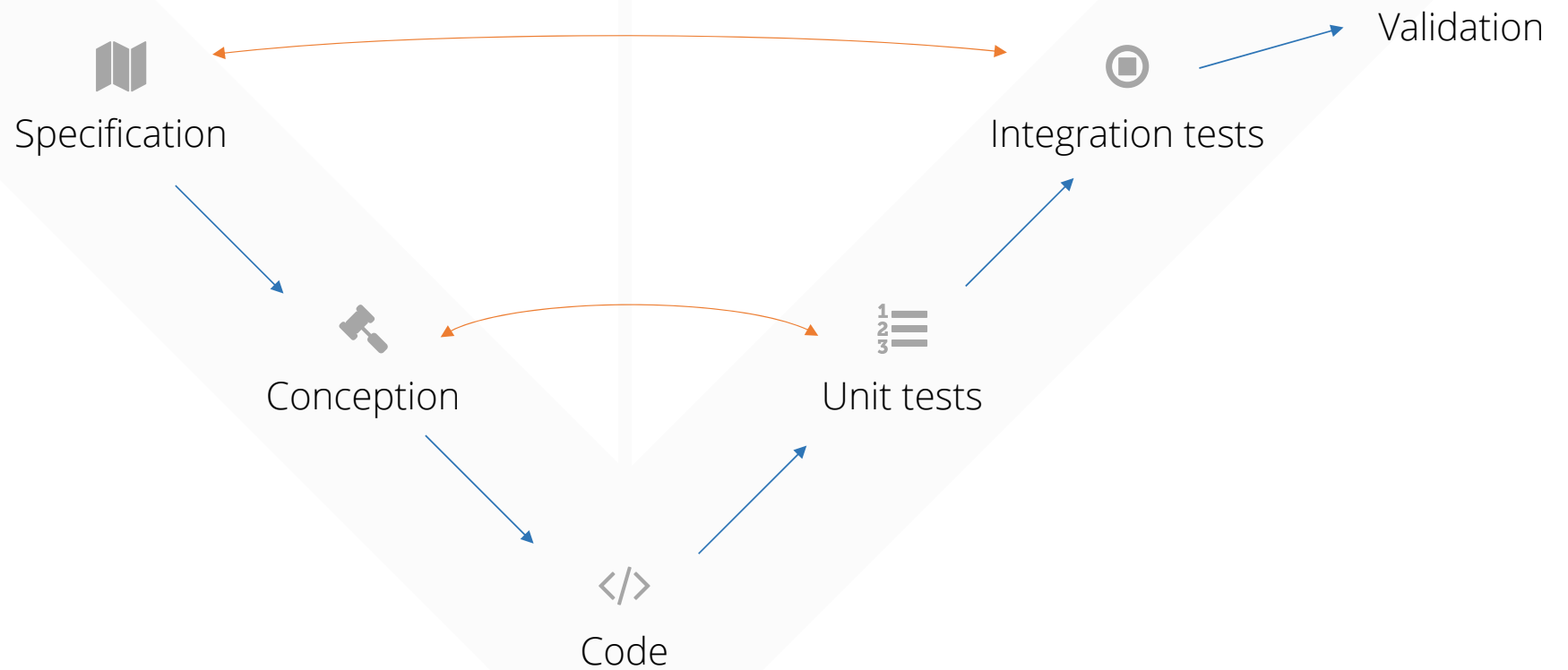
80 million passengers in 2009 

Other use cases

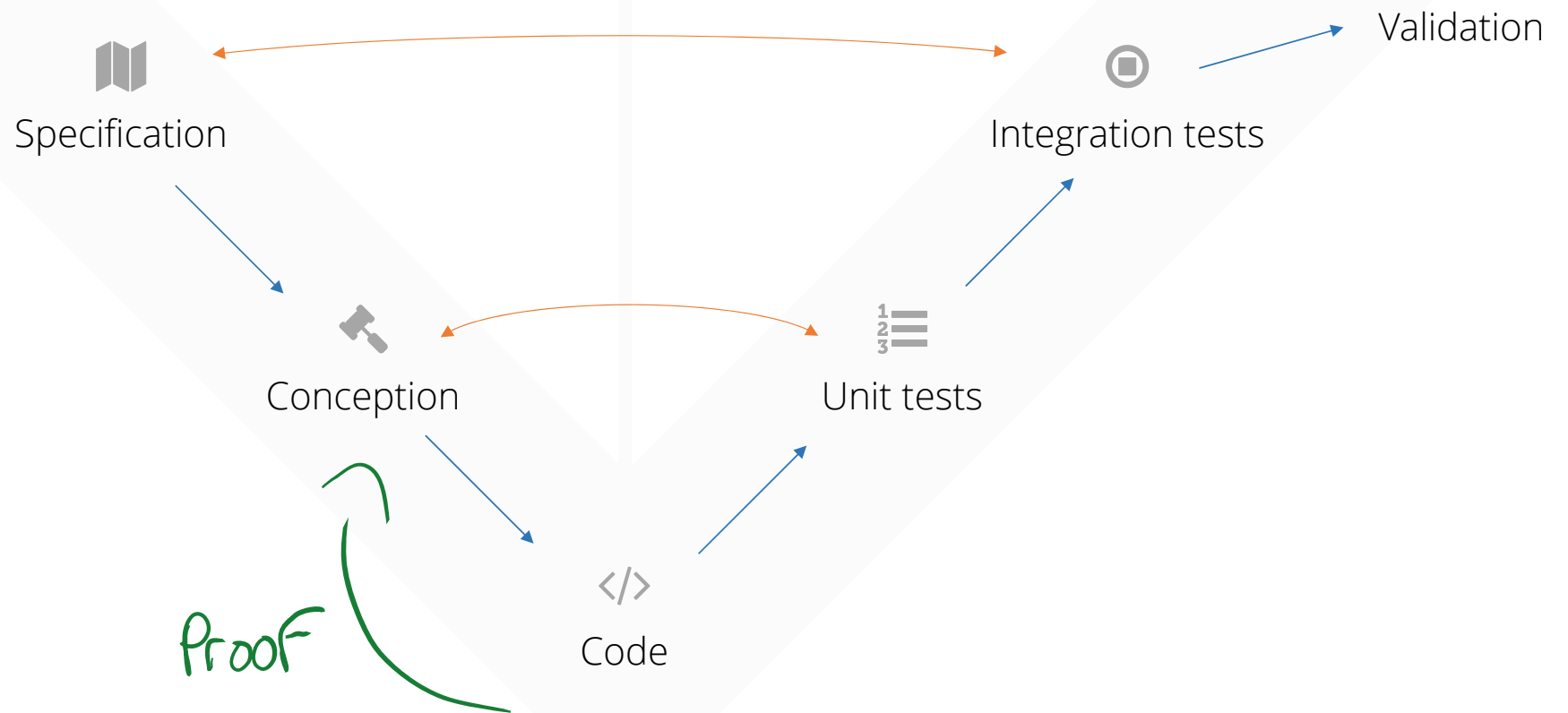
- Peugeot cars: formalization of sub systems (lights system, airbags, motor) to help building diagnostic tools
- Modeling of tasks scheduling from the software controlling the stage separations of Ariane rocket
- Protocol study
- JavaCard runtime formalization
 - Java runtime for *smartcard*
 - └─ Provide *safe*: authentication, data storage, application processing

Developing in B

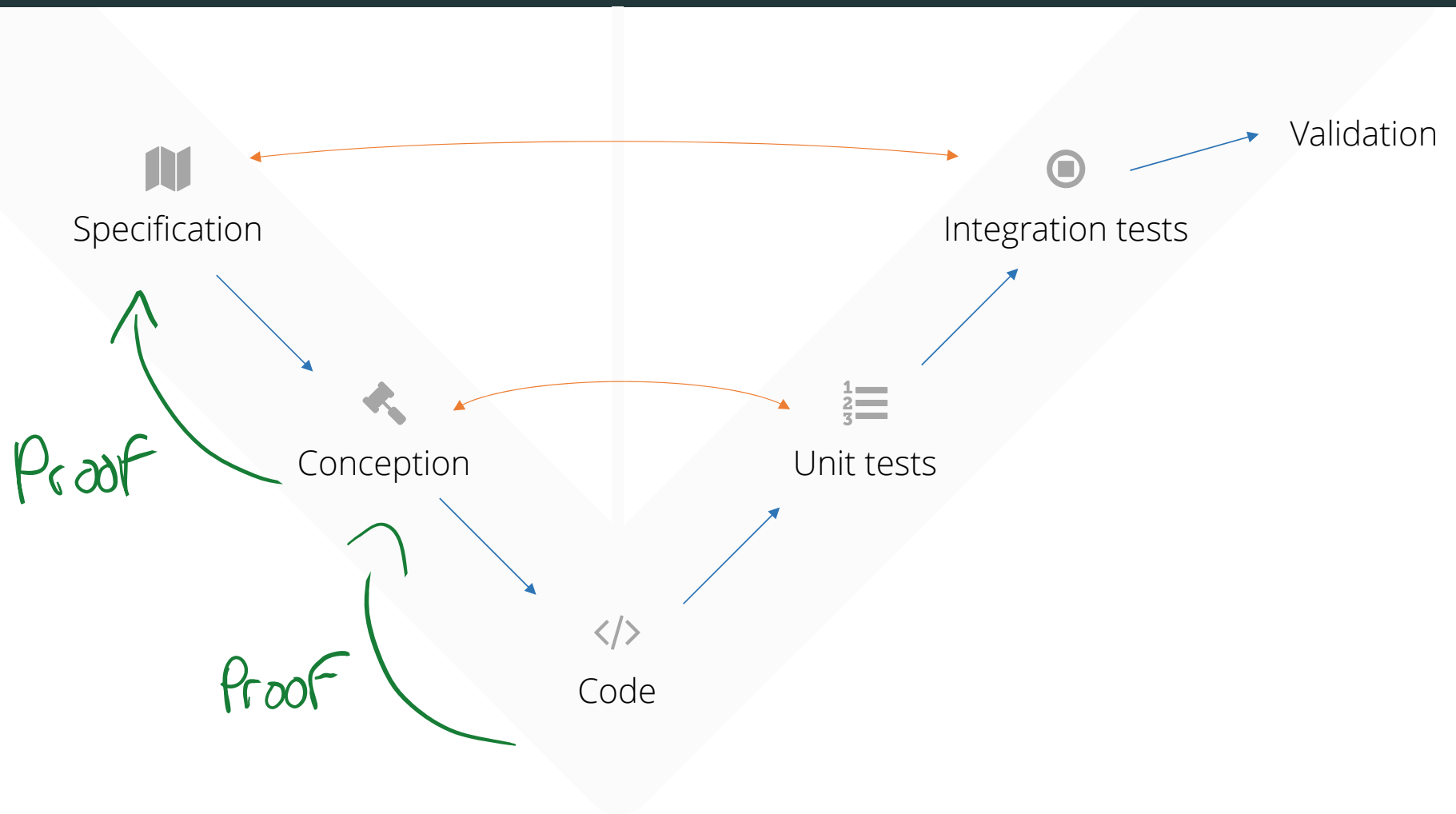
B method development



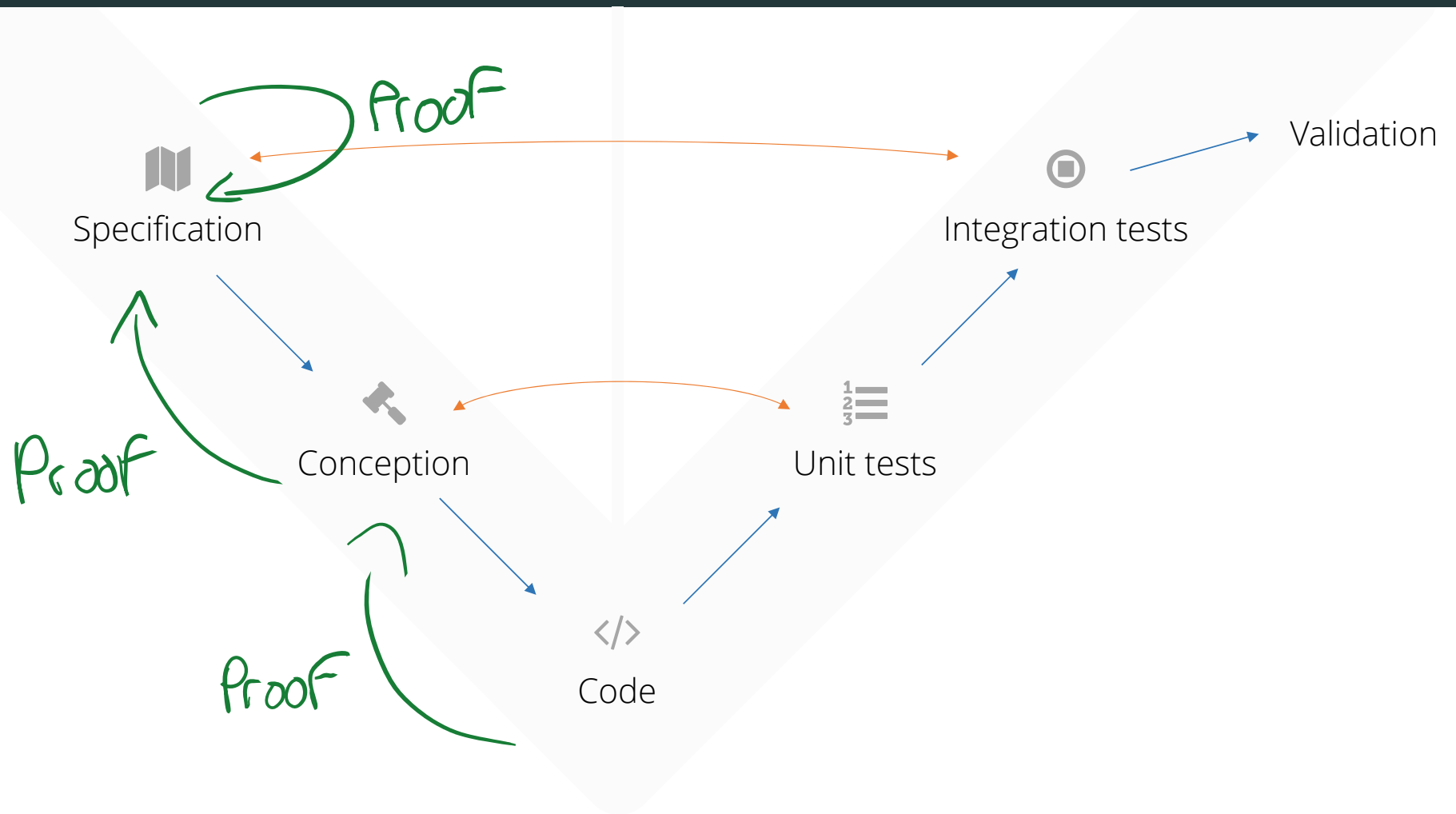
B method development



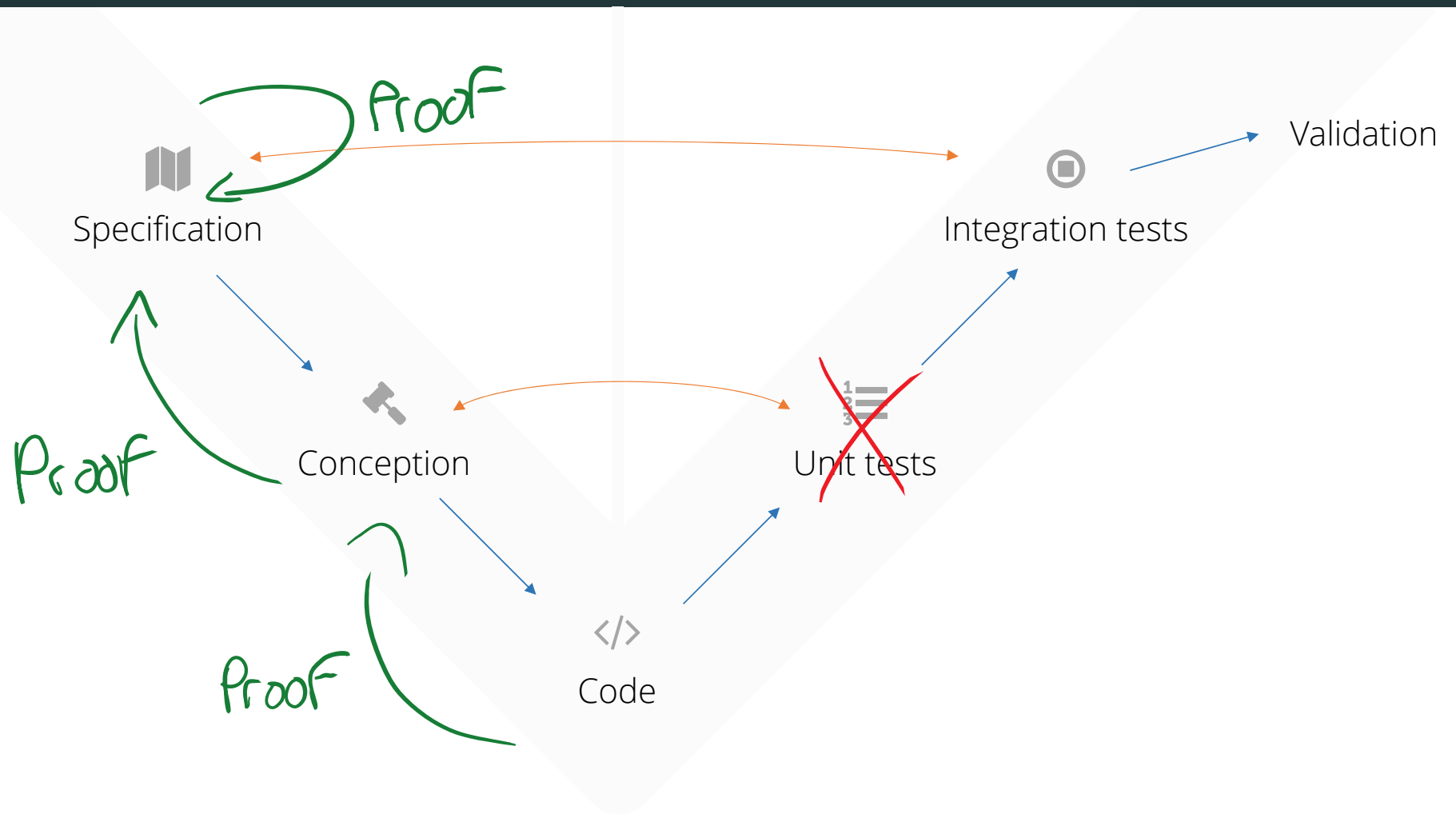
B method development



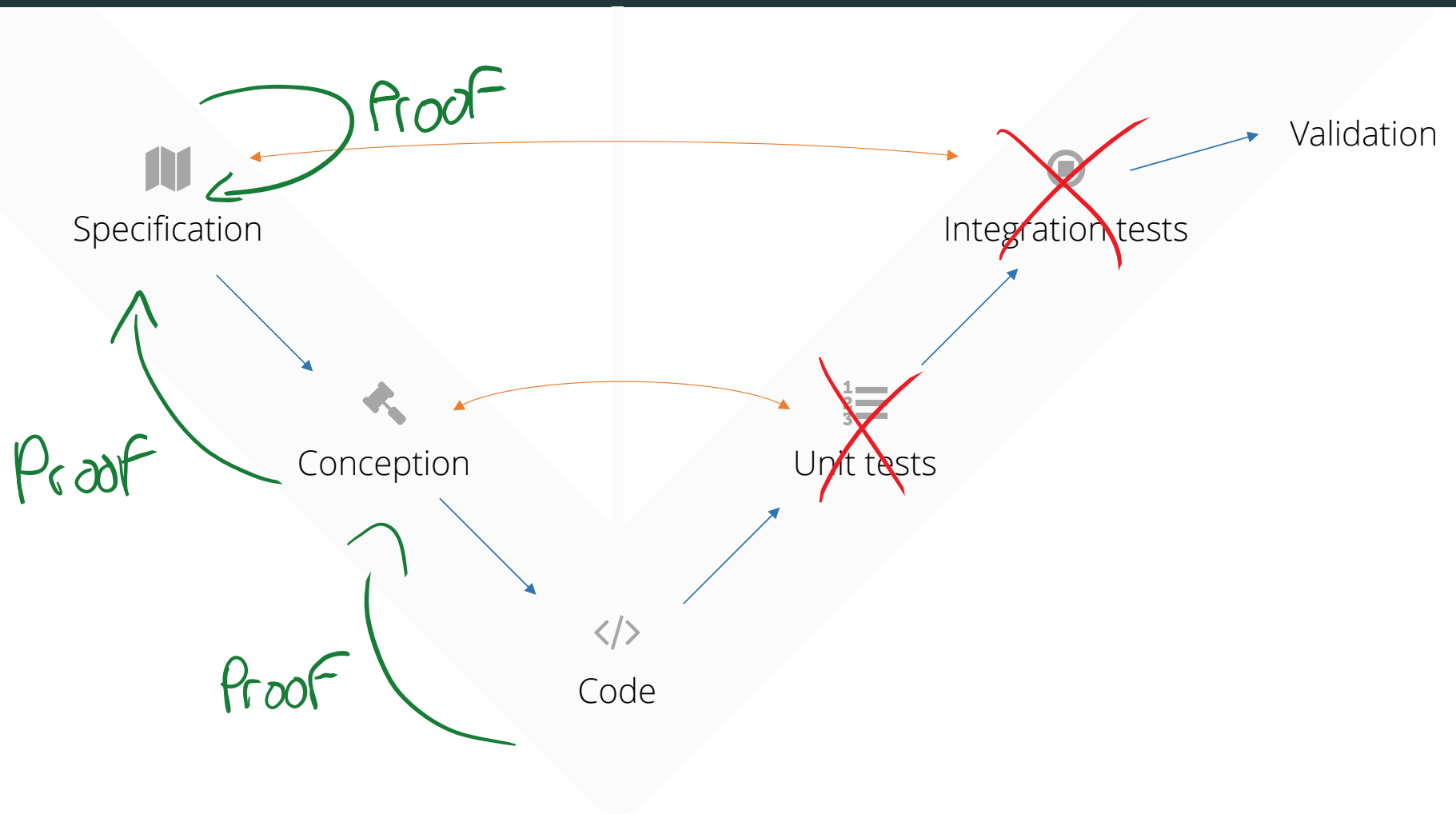
B method development



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

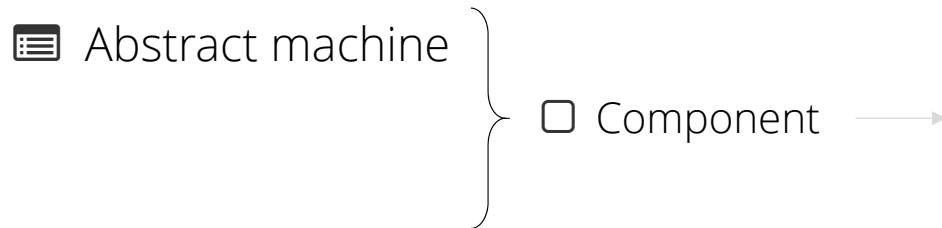
Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

📄 Module: *modelling of a sub system*



Static part

Definition of: Variables, constants, sets
List of invariants

Dynamic part

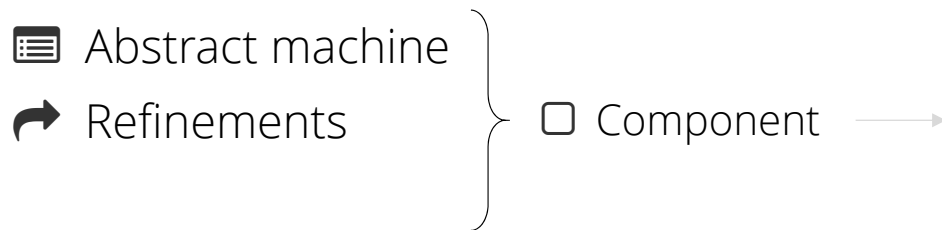
Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

📄 Module: *modelling of a sub system*



Static part

Definition of: Variables, constants, sets
List of invariants

Dynamic part

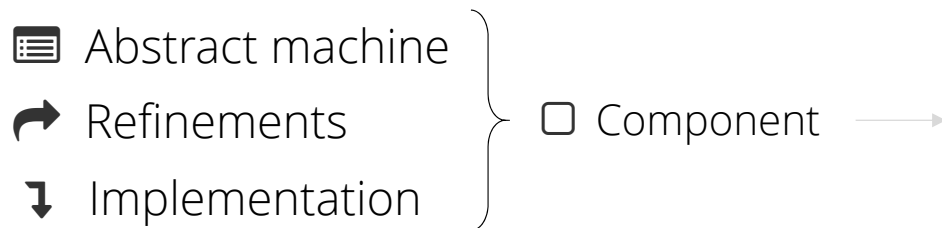
Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

📄 Module: *modelling of a sub system*



Static part

Definition of: Variables, constants, sets
List of invariants

Dynamic part

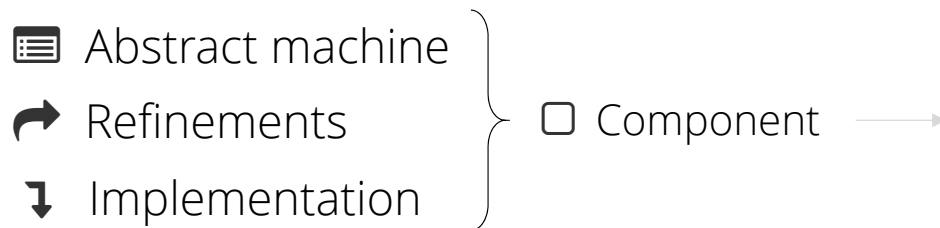
Initialize variables
Define operations on variables

Proof

Static part coherence
Initializing preserve invariants
Operations preserve invariants

How does method B works

📄 Module: *modelling of a sub system*



↓

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

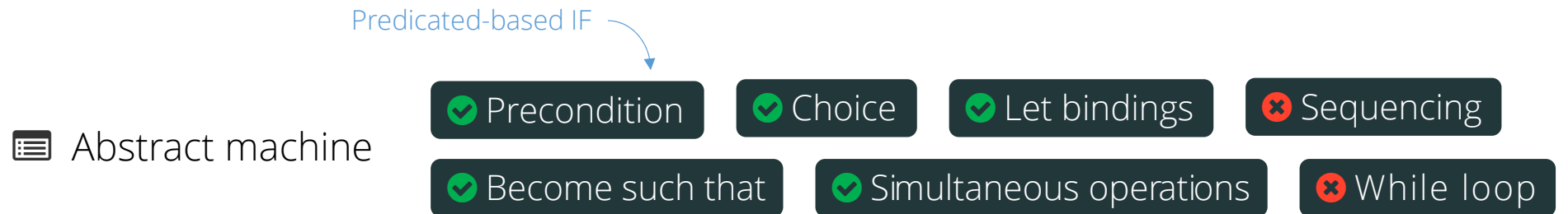
✗ While loop

↩ Refinements

⌵ Implementation

Substitutions

Machine, refinement, implementation

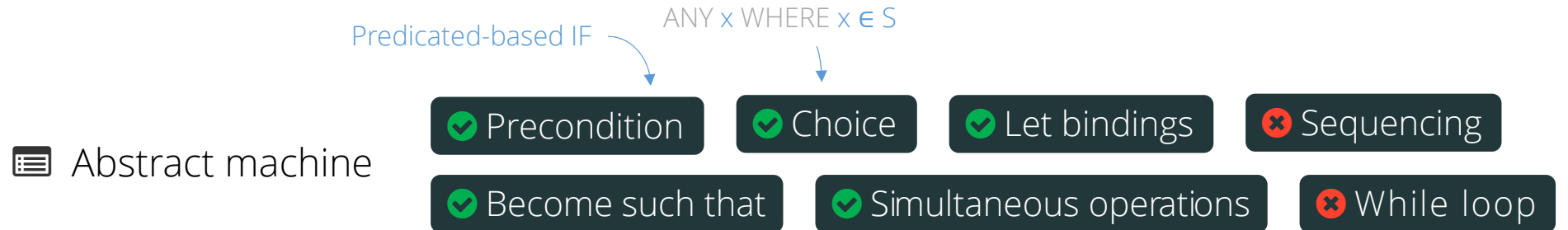


➡ Refinements

⬇ Implementation

Substitutions

Machine, refinement, implementation

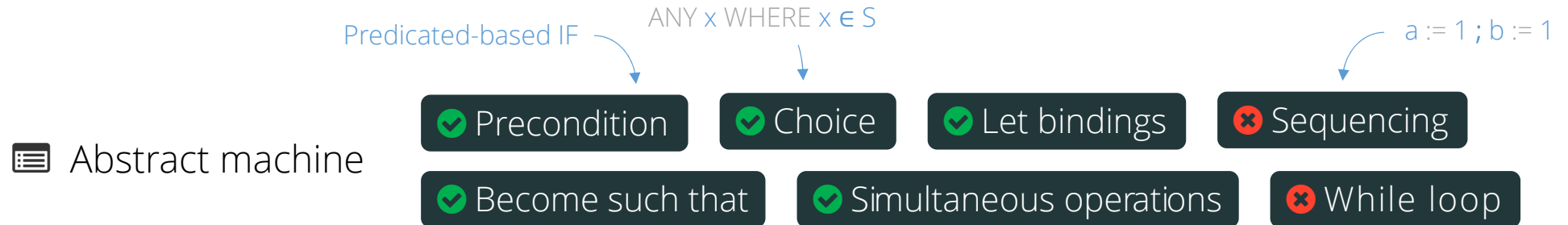


➡ Refinements

⤵ Implementation

Substitutions

Machine, refinement, implementation

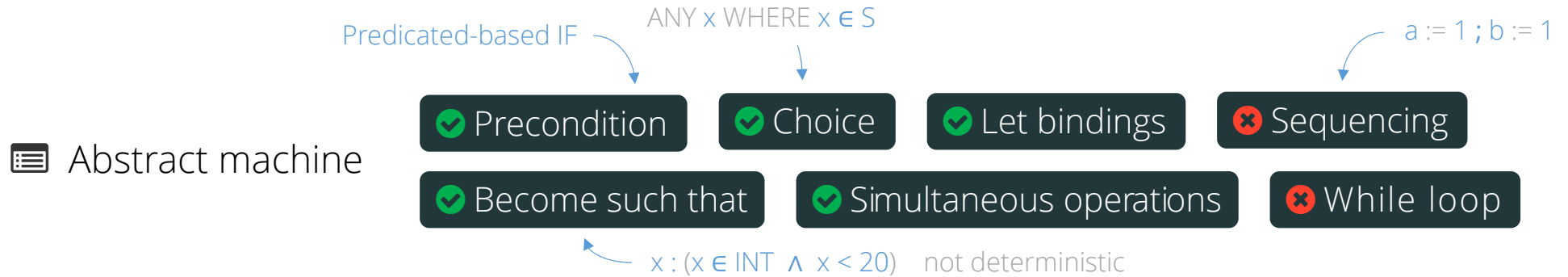


↪ Refinements

⌵ Implementation

Substitutions

Machine, refinement, implementation

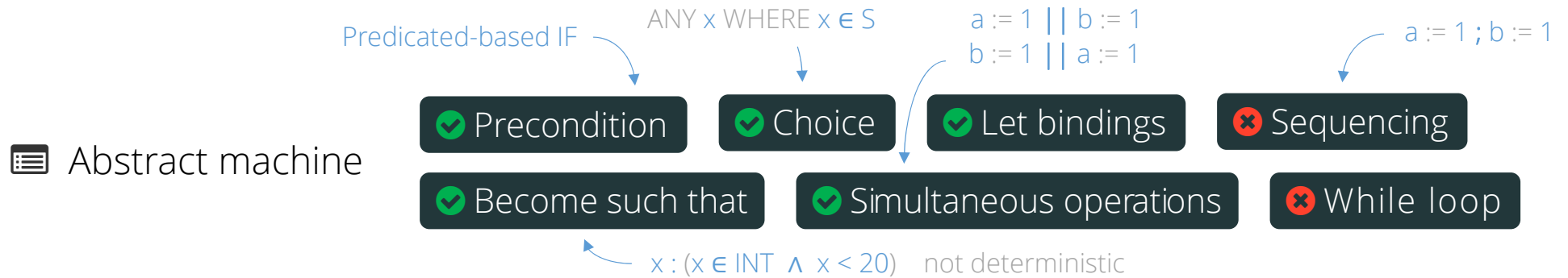


↪ Refinements

⌵ Implementation

Substitutions

Machine, refinement, implementation



➡ Refinements

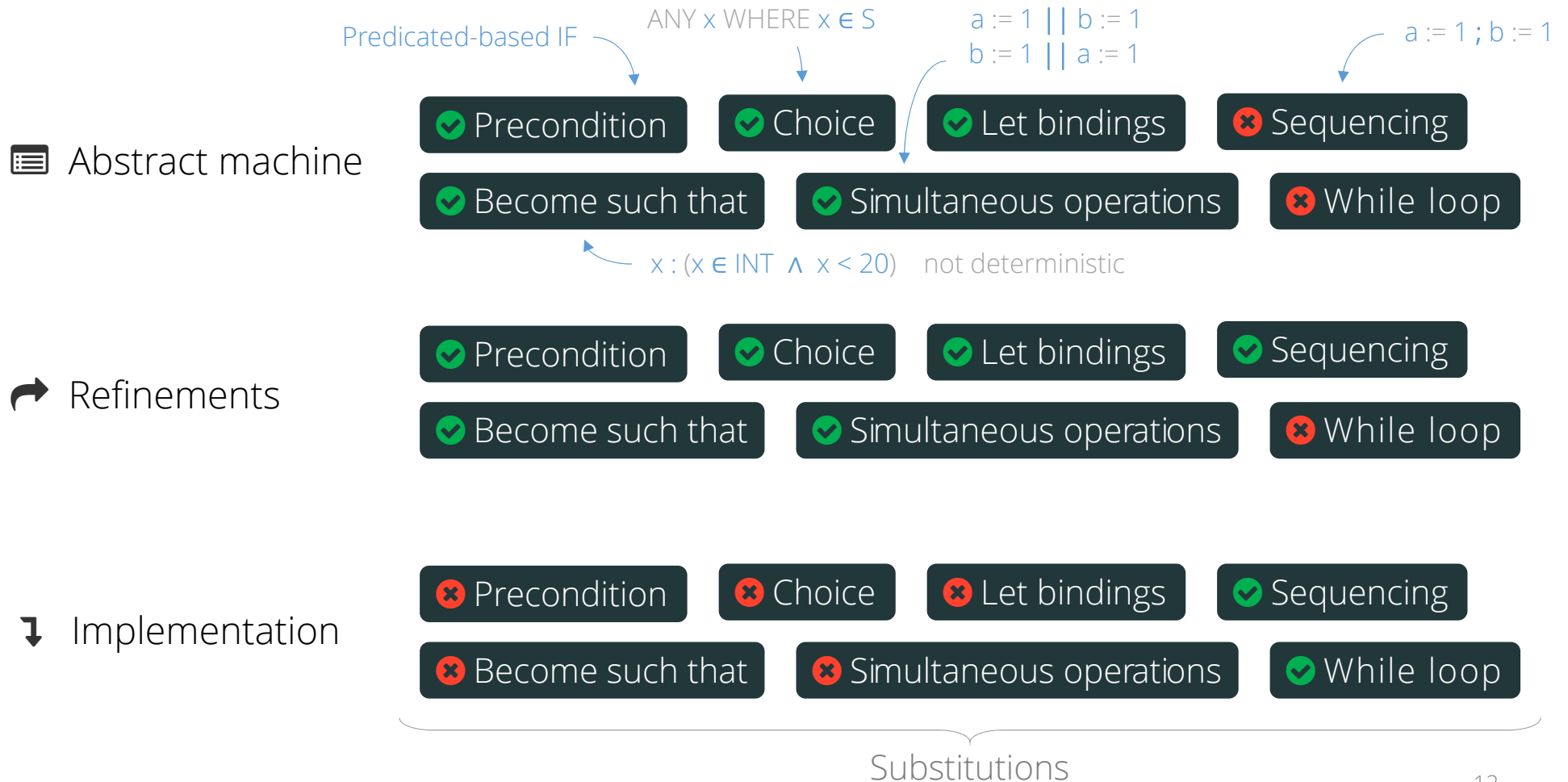
⤵ Implementation

Substitutions

Machine, refinement, implementation

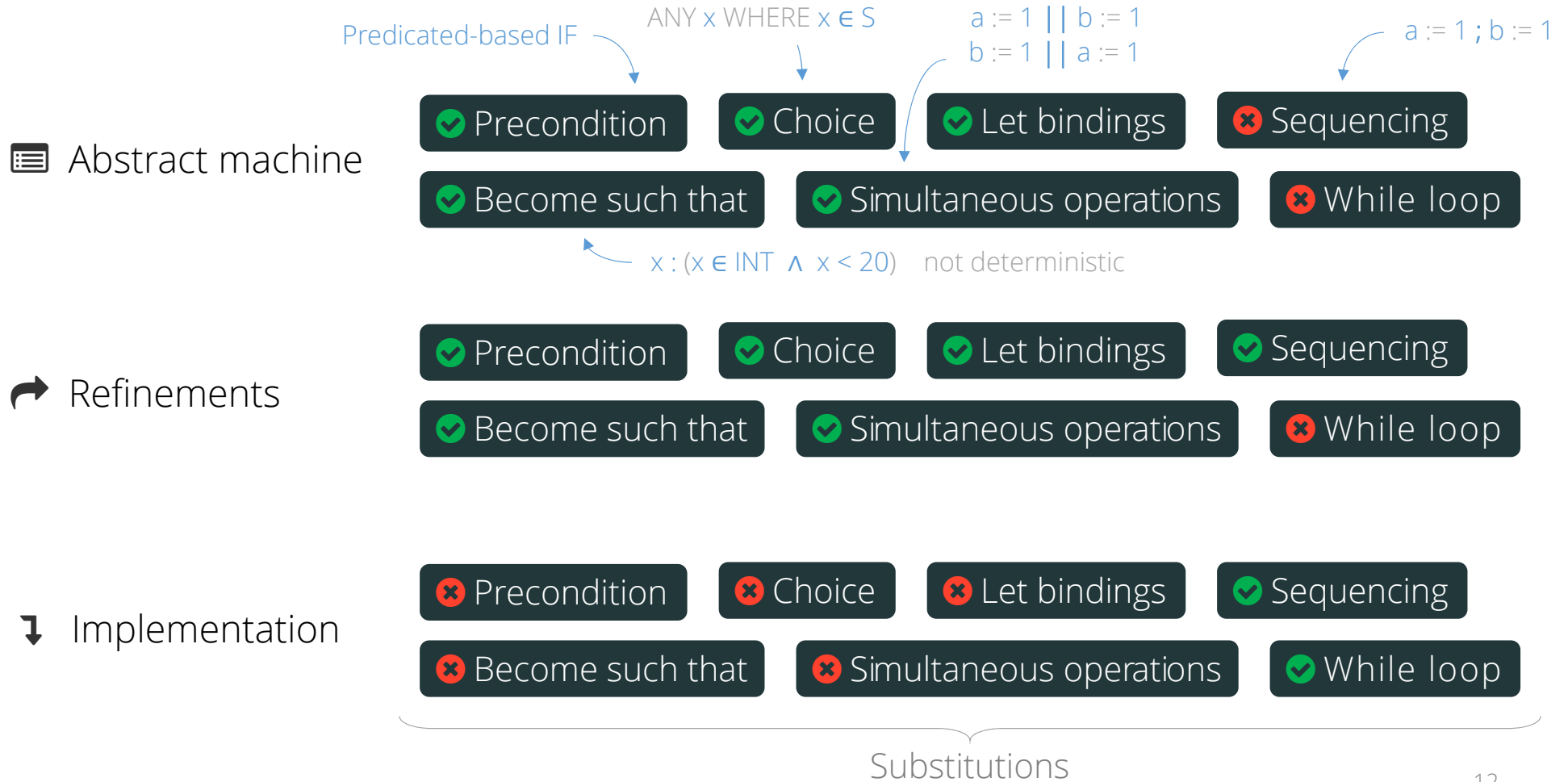


Machine, refinement, implementation



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

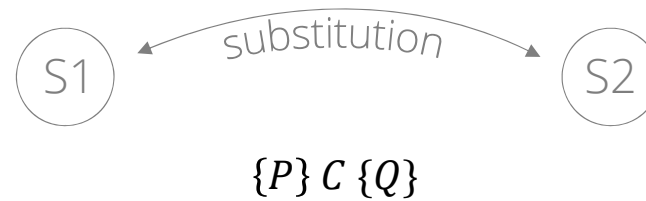
State oriented



B language

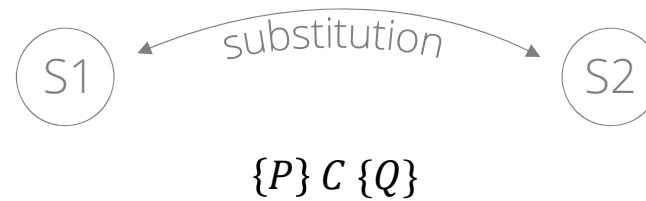
State oriented

Hoare logic



B language

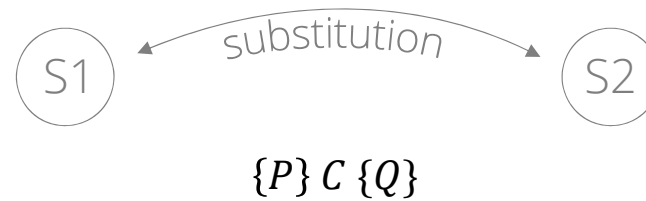
State oriented
Hoare logic



$\{true\} \ y := 3 \ \{y = 3\}$
 $\{x > 2\} \ x := x * x \ \{x > 4\}$

B language

State oriented
Hoare logic

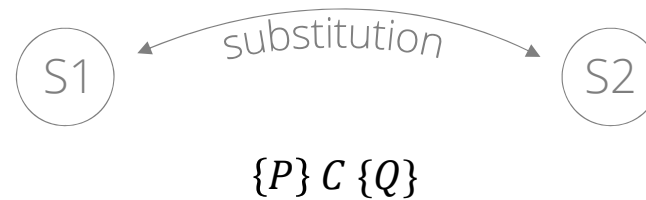


$\{true\} \ y := 3 \ \{y = 3\}$
 $\{x > 2\} \ x := x * x \ \{x > 4\}$

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

B language

State oriented
Hoare logic



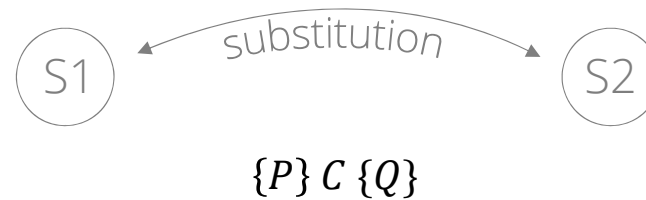
$\{true\} \ y := 3 \ \{y = 3\}$
 $\{x > 2\} \ x := x * x \ \{x > 4\}$

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

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

B language

State oriented
Hoare logic



$\{true\} \ y := 3 \ \{y = 3\}$
 $\{x > 2\} \ x := x * x \ \{x > 4\}$

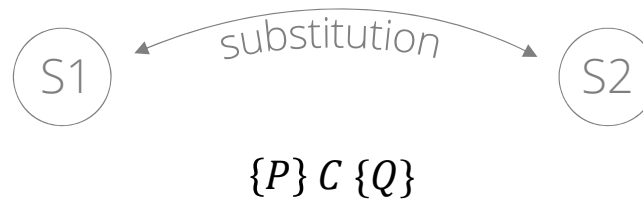
$$\frac{}{\{P[E/x]\} \ x := E \ \{P\}} \text{ASSIGNMENT}$$

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

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

B language

State oriented
Hoare logic



$\{true\} \ y := 3 \ \{y = 3\}$
 $\{x > 2\} \ x := x * x \ \{x > 4\}$

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

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

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

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

B language

State oriented

Hoare logic



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

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

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

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

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

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

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

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

B language

B language

Arithmetic

B language

Arithmetic

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

B language

Arithmetic

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

B language

Arithmetic

$+$ $-$ $\div$ x^y $\times$ $<$ $>$ $\leq$ $\geq$ $\mathbb{Z}$ $\mathbb{N}$ Π Σ

B language

Arithmetic

$+$ $-$ $\div$ x^y $\times$ $<$ $>$ $\leq$ $\geq$ $\mathbb{Z}$ $\mathbb{N}$ Π Σ

Functions and relations

B language

Arithmetic

$+$ $-$ $\div$ x^y $\times$ $<$ $>$ $\leq$ $\geq$ $\mathbb{Z}$ $\mathbb{N}$ Π Σ

Functions and relations

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

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightharpoonup$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightharpoonup$ R^+ $\mapsto$
 $\mapsto$ $\leftrightarrow$ $\rightharpoonup$ $\otimes$ $\mapsto$ r^{-1}

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightrightarrows$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightrightarrows$ R^+ $\mapsto$
 $\mapsto$ $\leftrightarrow$ $\rightrightarrows$ $\otimes$ $\mapsto$ r^{-1}

Sets

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightharpoonup$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightharpoonup$ R^+ $\rightarrow$
 $\mapsto$ $\leftrightarrow$ $\mapsto$ $\otimes$ $\mapsto$ r^{-1}

Sets

Set comprehension, generalized union & intersections

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightharpoonup$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightharpoonup$ R^+ $\mapsto$
 $\rightarrow$ $\leftrightarrow$ $\rightharpoonup$ $\otimes$ $\mapsto$ r^{-1}

Sets

Set comprehension, generalized union & intersections

Records

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightrightarrows$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightrightarrows$ R^+ $\mapsto$
 $\rightarrow$ $\leftrightarrow$ $\rightrightarrows$ $\otimes$ $\mapsto$ r^{-1}

Sets

Set comprehension, generalized union & intersections

Records

Trees

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightrightarrows$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightrightarrows$ R^+ $\rightarrow$
 $\mapsto$ $\leftrightarrow$ $\rightrightarrows$ $\otimes$ $\mapsto$ r^{-1}

Sets

Set comprehension, generalized union & intersections

Records

Trees

Sequences

B language

Arithmetic

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

Functions and relations

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

$\rightarrow$ λ $\Leftarrow$ $\rightrightarrows$ R^* $\triangleright$
 $\mapsto$ $\Leftarrow$ $\triangleleft$ $\rightrightarrows$ R^+ $\rightarrow$
 $\mapsto$ $\leftrightarrow$ $\rightrightarrows$ $\otimes$ $\mapsto$ r^{-1}

Sets

Set comprehension, generalized union & intersections

Records

Trees

Sequences

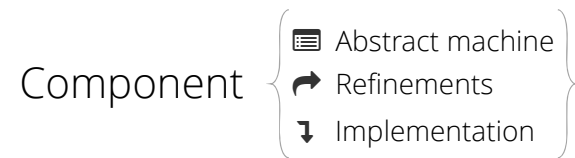
No algebraic data types

Proofs with B

How does B method handles proofs?



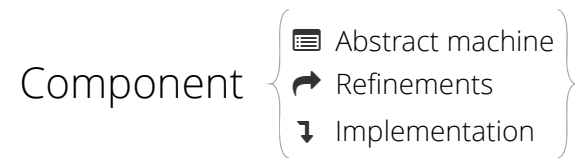
How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

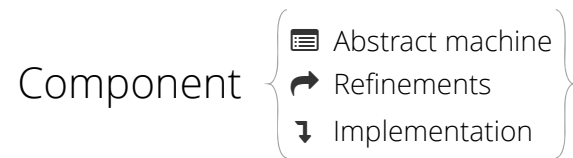
CONSTRAINTS

input1 ∈ INT ∧ input2 ∈ INT ...

CONSTANTS

cst1, cst2, ...

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 ∈ INT ∧ input2 ∈ INT ...

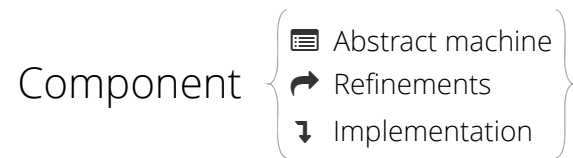
CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

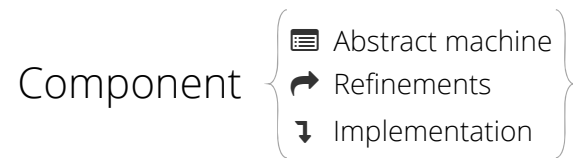
VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

ASSERTIONS

predicate1 $\wedge$ predicate2 $\wedge$...

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

ASSERTIONS

predicate1 $\wedge$ predicate2 $\wedge$... *lemmas*

How does B method handles proofs?



MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

Prove all predicates {

ASSERTIONS

predicate1 $\wedge$ predicate2 $\wedge$... *lemmas*

How does B method handles proofs?



Prove all predicates {

ASSERTIONS
predicate1 $\wedge$ predicate2 $\wedge$... *lemmas*

INITIALISATION
var1 := expr || var2 := expr

MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

ASSERTIONS

predicate1 $\wedge$ predicate2 $\wedge$...

INITIALISATION

var1 := expr || var2 := expr

How does B method handles proofs?



Prove all predicates {
For each initialization, prove invariant conservations {

MACHINE

Name(input1, input2, ...)

CONSTRAINTS

input1 $\in$ INT $\wedge$ input2 $\in$ INT ...

CONSTANTS

cst1, cst2, ...

VARIABLES

var1, var2, ...

INVARIANT

var1 + var2 $\in$ {x . $X^{**}(1/2) \in \mathbb{Z}$ } $\wedge$...

ASSERTIONS

predicate1 $\wedge$ predicate2 $\wedge$... *lemmas*

INITIALISATION

var1 := expr || var2 := expr

How does B method handles proofs?



Prove all predicates {

For each initialization, prove invariant conservations {

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*

INITIALISATION
var1 := expr || var2 := expr

OPERATIONS
varOutput1 ← fun1(i1, i2, ...) =
...
varOutput2 ← fun2(i1, i2, ...) =
...

How does B method handles proofs?



Prove all predicates {

For each initialization, prove invariant conservations {

Show each operation conserve invariants {

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*

INITIALISATION
var1 := expr || var2 := expr

OPERATIONS
varOutput1 ← fun1(i1, i2, ...) =
...
varOutput2 ← fun2(i1, i2, ...) =
...

How does B method handles proofs?



☆ Proof Obligations

Prove all predicates

For each initialization, prove
invariant conservations

Show each operation
conserve invariants

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

INITIALISATION

var1 := expr || var2 := expr

OPERATIONS

varOutput1 ← fun1(i1, i2, ...) =

...

varOutput2 ← fun2(i1, i2, ...) =

...

Proving with B method: interactive only

Atelier B

Atelier B View Workspace Project Component Help

Workspace: SeatReservation (OK|OK|99|61|38%)

Classical view

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Reservation	OK	OK	12	12	0	-
Reservation_i	OK	OK	36	22	14	-
Reservation_r	OK	OK	51	4	47	-

Tasks

Hide Finished tasks

Project	Component	Action	Status	Messages	Series
SeatRe...	Reservati...	Po	Finished	36 proof ...	local
SeatRe...	Reservati...	Po	Finished	End of Pr...	local

Errors

Errors (0) Warnings (0) Multi-Line messages

Message

No errors

Proving with B method: interactive only

Atelier B

Atelier B View Workspace Project Component Help

Workspace... x

Filter

SeatReservation (OK|OK|99|61|38%)

Classical view

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Reservation	OK	OK	12	12	0	-
Reservation_i	OK	OK	36	22	14	-
Reservation_r	OK	OK	51	4	47	-

Tasks

Hide Finished tasks

Project	Component	Action	Status	Messages	Series
SeatRe...	Reservati...	Po	Finished	36 proof ...	local
SeatRe...	Reservati...	Po	Finished	End of Pr...	local

Errors

Errors (0) Warnings (0) Multi-Line messages

Message

No errors

Proving with B method: interactive only

Atelier B

Atelier B View Workspace Project Component Help

Workspace... x

Filter

SeatReservation (OK|OK|99|61|38%)

Classical view

Filter Clear

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Reservation	OK	OK	12	12	0	-
Reservation_i	OK	OK	36	22	14	-
Reservation_r	OK	OK	51	4	47	-

Tasks

Hide Finished tasks

Project	Component	Action	Status	Messages	Series
SeatRe...	Reservati...	Po	Finished	36 proof ...	local
SeatRe...	Reservati...	Po	Finished	End of Pr...	local

Errors

Errors (0) Warnings (0) Multi-Line messages

Message

No errors

Proving with B method: interactive only

Atelier B

Atelier B View Workspace Project Component Help

Workspace... x

Filter

Classical view

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Reservation	OK	OK	12	12	0	-
Reservation_i	OK	OK	36	22	14	-
Reservation_r	OK	OK	51	4	47	-

Proofs tasks : 3 forces

Tasks

Hide Finished tasks

Project	Component	Action	Status	Messages	Series
SeatRe...	Reservati...	36	Finished	36 proof ...	local
SeatRe...	Reservati...	51	Finished	End of Pr...	local

Errors

Errors (0) Warnings (0) Multi-Line messages

Message

No errors

Proving with B method: interactive only

The screenshot shows the Atelier B IDE interface. The main window displays the 'Classical view' of the 'SeatReservation' project. A table titled 'Proof Obligations' is shown, with columns: Component, TypeChecked, POs Generated, Proof Obligations, Proved, Unproved, and B0 Checked. The table contains three rows: 'Reservation' (12 POs, 12 Proved, 0 Unproved), 'Reservation_i' (36 POs, 22 Proved, 14 Unproved), and 'Reservation_r' (51 POs, 4 Proved, 47 Unproved). The 'Reservation_i' row is highlighted, and its 'Proved' and 'Unproved' values are circled in green. A handwritten note 'Proofs tasks : 3 forces (automatic)' with an arrow points to the 'Tasks' panel at the bottom. The 'Tasks' panel shows a list of tasks, including '36 proof ...' and 'End of Pr ...'. The 'Errors' panel at the bottom right shows 'No errors'.

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
Reservation	OK	OK	12	12	0	-
Reservation_i	OK	OK	36	22	14	-
Reservation_r	OK	OK	51	4	47	-

Proofs tasks : 3 forces (automatic)

Project	Component	Action	Status	Messages	Series
SeatRe...	Reservati...	36	Finished	36 proof ...	local
SeatRe...	Reservati...	51	Finished	End of Pr ...	local

Proving with B method: interactive only

The screenshot displays the Atelier B Prover interface for the 'Reservation_i - SeatReservation' project. The interface is divided into several panes:

- Workspace:** Shows the project structure with components 'Reservation', 'Reservation_i', and 'Reservation_r'. 'Reservation_i' is selected.
- Classical view:** A table showing the status of components.
- Tasks:** A table listing tasks and their completion status.
- Situation:** A pane showing the current state of the proof, including 'POs recently proved' and 'All POs'.
- Proof:** A pane showing the current proof obligation and the next step to be proved.
- Initial proof obligation:** A pane showing the initial proof obligation.
- Search hypothesis result:** A pane showing the results of the search for hypotheses.

The 'Situation' pane shows the following components and their status:

Component	TypeChecked
Reservation	OK
Reservation_i	OK
Reservation_r	OK

The 'Proof' pane shows the following proof obligation:

```
Force(0)
Next
pr
```

The 'Initial proof obligation' pane shows the following initial proof obligation:

```
btrue
=>
(1 .. nmax) * {FALSE} : 1 .. nmax --> BOOL
```

The 'Search hypothesis result' pane shows the following search hypothesis result:

```
"Refinement is correct" &
"Check invariant ((state$1) : (((1) .. (nmax))) --> (BOOL)))"
=>
(1..nmax)*{FALSE}: 1..nmax --> BOOL
```

An arrow points from the 'Tasks' pane to the 'Proof' pane, indicating the current task being proved.

Proving with B method: interactive only

The screenshot displays the Atelier B Prover interface for a project named "Reservation_i - SeatReservation". The interface is divided into several panes:

- Workspace:** Shows a tree view of the project components. The "Reservation_i" component is selected, and its "Classical view" is displayed. The table below shows the status of the components:

Component	TypeChecked
Reservation	OK
Reservation_i	OK
Reservation_r	OK

- Tasks:** A table showing the progress of tasks. The "SeatRe..." task is marked as "Finished".
- Situation:** A pane showing the current state of the proof. It includes a list of "POs recently proved" and "All POs". The "PO1" is listed under "All POs".
- Proof:** A pane showing the current proof obligation. The "Force(0)" obligation is selected, and the next step is "pr".
- Initial proof obligation:** A pane showing the initial proof obligation, which is "btrue".
- Search hypothesis result:** A pane showing the search hypothesis result, which is "btrue".

The main proof area displays the following code:

```
"Refinement is correct" &  
"Check invariant ((state$1): (((1) .. (nmax))) --> (BOOL)))"  
=>  
(1..nmax) * {FALSE}: 1..nmax --> BOOL
```

How does B method internally prove things?

First, expressions are normalized

Expression	Normal Form
$n > m$	$m + 1 \leq n$
$m < n$	$m + 1 \leq n$
$a \leq b$	$(a \Rightarrow b) \wedge (b \Rightarrow a)$
$a <: b$	$a : POW(b)$
$a <<: b$	$a : POW(b) \wedge \neg(a = b)$
$a / : b$	$\neg(a : b)$
$a / = b$	$\neg(a = b)$
$a / <: b$	$\neg(a : POW(b))$
$a / <<: b$	$a : POW(b) \Rightarrow a = b$
$a : NATURAL$	$a : INTEGER \wedge 0 \leq a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E) / \setminus (NATURAL - \{0\} + - >> E)$
$\langle \rangle$	$\{\}$
$\{x, y\}$	$\{x\} \setminus \{y\}$
$\{x P\}$	$SET(x).P$

How does B method internally prove things?

First, expressions are normalized

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

Expression	Normal Form
$n > m$	$m + 1 \leq n$
$m < n$	$m + 1 \leq n$
$a \leq b$	$(a \leq b) \wedge (b \leq a)$
$a \leq b$	$a : POW(b)$
$a \leq b$	$a : POW(b) \wedge \neg(a = b)$
a / b	$\neg(a : b)$
$a / = b$	$\neg(a = b)$
$a / \leq b$	$\neg(a : POW(b))$
$a / \leq b$	$a : POW(b) \Rightarrow a = b$
$a : NATURAL$	$a : INTEGER \wedge 0 \leq a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E) / \setminus (NATURAL - \{0\} + - > E)$
$\langle \rangle$	$\{\}$
$\{x, y\}$	$\{x\} \setminus \{y\}$
$\{x P\}$	$SET(x).P$

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
$n > m$	$m + 1 \leq n$
$m < n$	$m + 1 \leq n$
$a \leq b$	$(a \Rightarrow b) \wedge (b \Rightarrow a)$
$a \leq b$	$a : POW(b)$
$a \leq b$	$a : POW(b) \wedge \neg(a = b)$
a / b	$\neg(a : b)$
$a / = b$	$\neg(a = b)$
$a / \leq b$	$\neg(a : POW(b))$
$a / \leq b$	$a : POW(b) \Rightarrow a = b$
$a : NATURAL$	$a : INTEGER \wedge 0 \leq a$
$NATURAL1$	$NATURAL - \{0\}$
$NAT1$	$NAT - \{0\}$
$FIN1(A)$	$FIN(A) - \{\{\}\}$
$POW1(A)$	$POW(A) - \{\{\}\}$
$seq1(A)$	$seq(A) - \{\{\}\}$
$iseq1(A)$	$iseq(A) - \{\{\}\}$
$perm(E)$	$iseq(E) / \setminus (NATURAL - \{0\} + - > E)$
$\langle \rangle$	$\{\}$
$\{x, y\}$	$\{x\} \setminus \{y\}$
$\{x P\}$	$SET(x).P$

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
;
    (bvr b(x)) &
    (x\ a)
=>
    #x.a == a
;
    (bvr b(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
;
    (bvr b(x)) &
    (x\a)
=>
    #x.a == a
;
    (bvr b(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

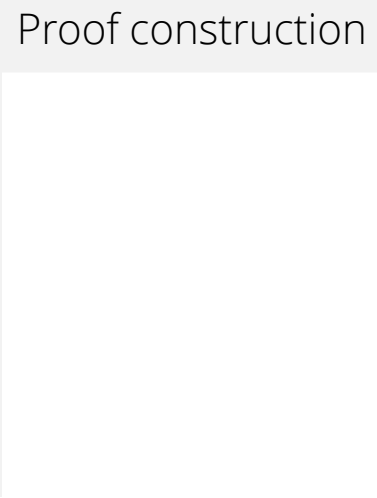
Search information Search rule / hypothesis / goal, show proof, reduce PO...

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*

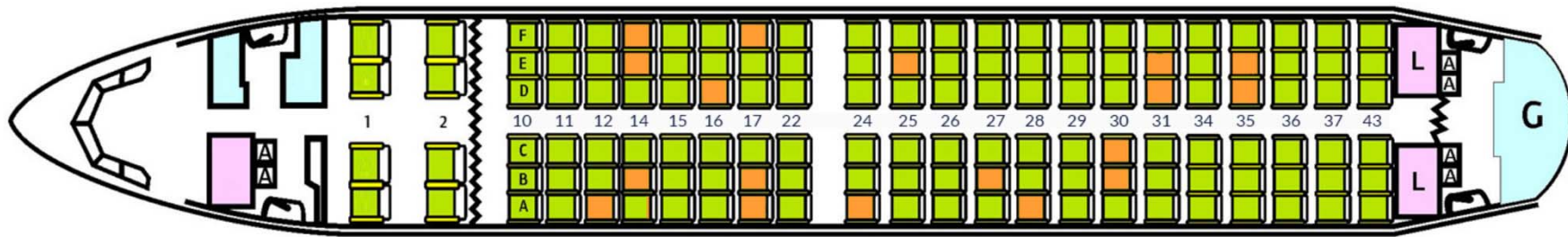
ct **fh** **dc** **se**

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?