Aligning SysML with the B method to Provide V&V for Systems Engineering

MoDeVVa 2012

Erwan Bousse David Mentré Benoît Combemale
Benoît Baudry

Mitsubishi Electric R&D Center Europe – IRISA/Inria (Rennes, France)

September 30, 2012

Idea, goals and choices

Problem DSMLs require to implement new V&V tools

$\Rightarrow$ time consuming and error prone task

Solution Translating DSMLs into existing formal languages

$\Rightarrow$ reuse existing V&V for formal languages

Additional requirements for this work:

- Traceability required between informal requirements and the modeled system, especially for safety properties
- Safety properties = invariants on states of the system

Our choices for this first approach: **SysML** and the **B method**

Background: studied languages

SysML (*Systems Modeling Language*)

- Structural and behavioral modeling for **systems engineering**
- Extends a subset of UML, graphical syntax
- Possible to enrich models with others languages, including:
  - → Alf (*Action Language for Foundational UML*)
  - → OCL (*Object Constraint Language*)

B method

- Software oriented **formal method**
- Based on set theory, Hoare logic, first order logic
- Uses abstract machines refined towards implementations
- Properties verified using theorem proving

How to use the B method for our approach?

How to use the B language at its “best”?
How to handle huge systems (scalability)?

B notions

Module	1 abstract machine (AM), 0+ refinements, 0-1 implementation
AM	Specification part – independent
Impl.	Implementation part – can use other modules

Two main possibilities:

1. **Purely abstract** modules linked by **includes**
2. **Developed** modules linked by **imports**

Industrial use of the B method

Problems with option 1

Abstract modules + includes = "one big module"

$\Rightarrow$ Scalability issues

Good/common industrial practices: option 2

Developed modules + imports = real **decomposition**

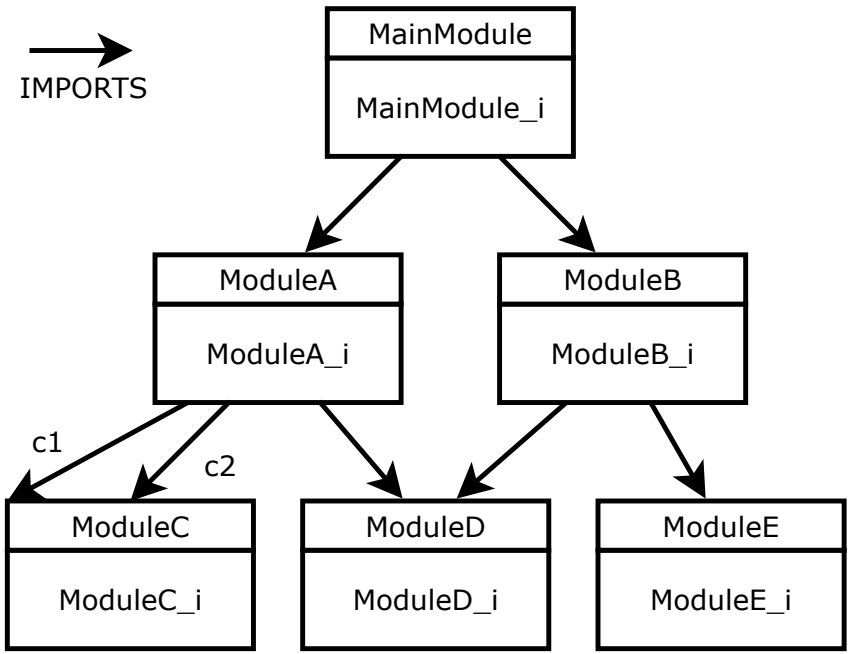
$\Rightarrow$ Better scalability: properties contained in subsystems

$\Rightarrow$ Can be compiled in C or Ada

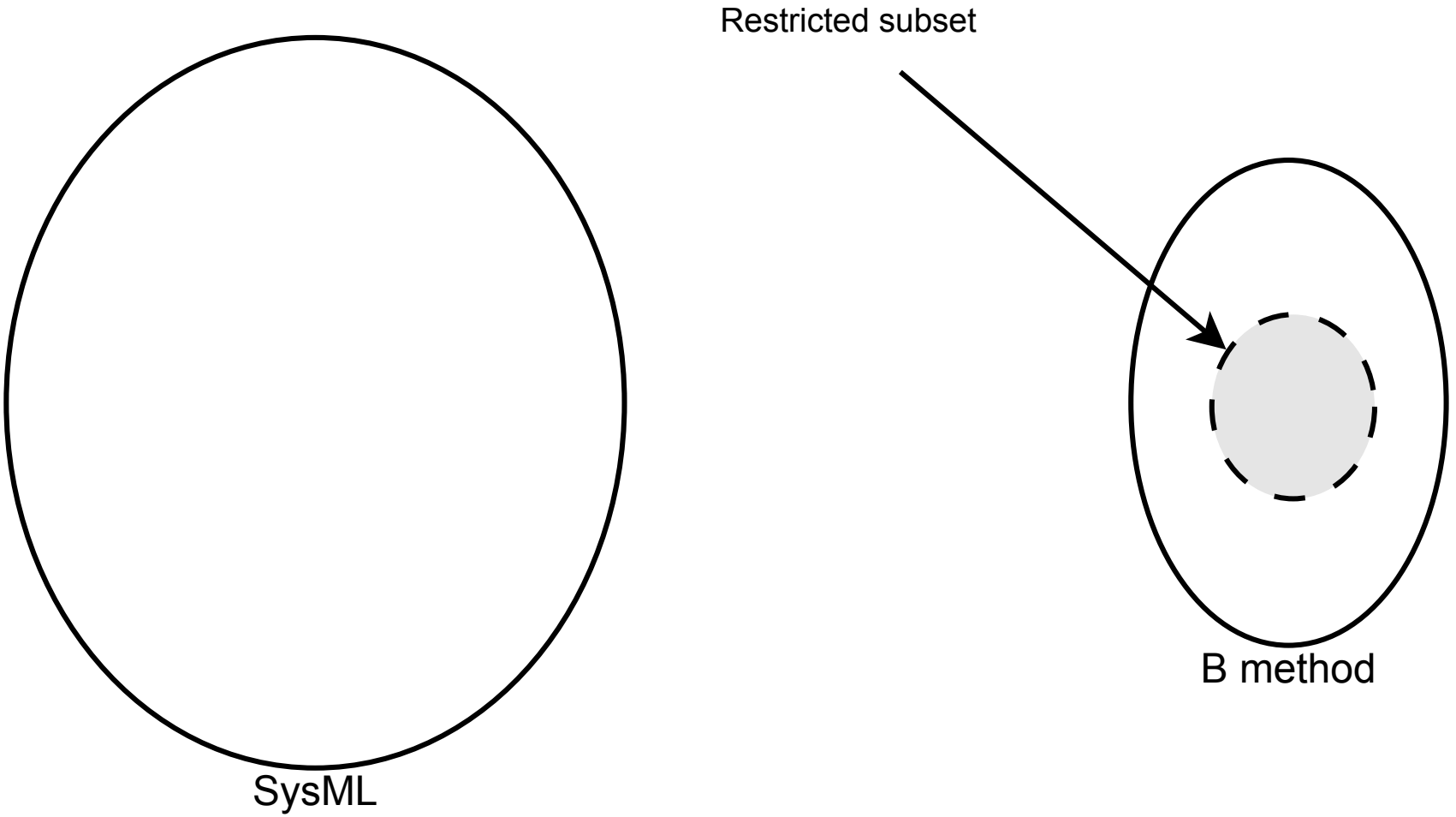
- Intermediate refinements rarely used
- A prevailing tool: Atelier-B

Our B subset for this first approach

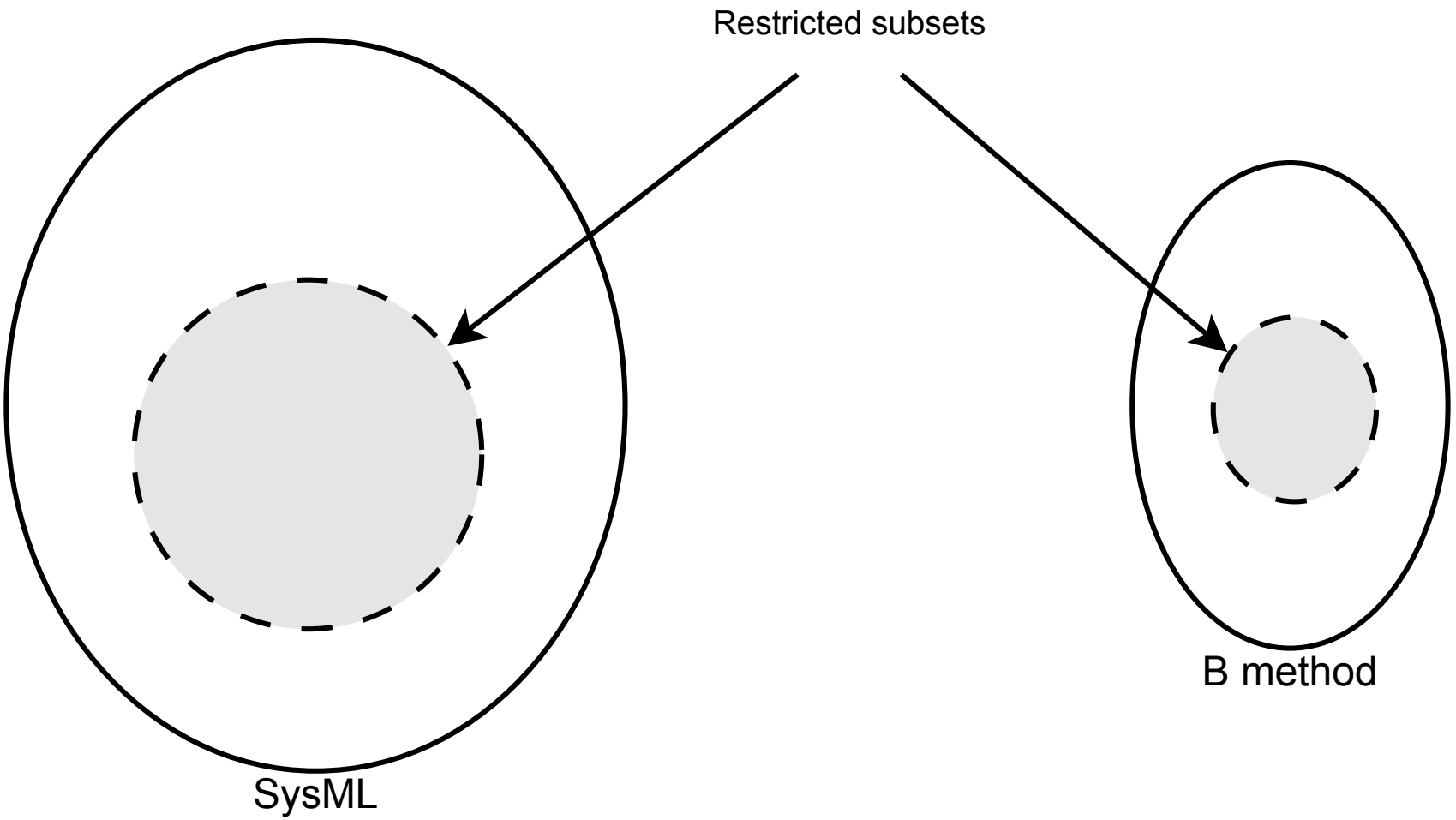
- **Developed modules** with 1 abstract machine and 1 implementation (no intermediate refinements)
- **Imports** links for instantiation and **sees** links for read accesses
- Primitive types only (boolean, integer and enumerations – no sets or relations)



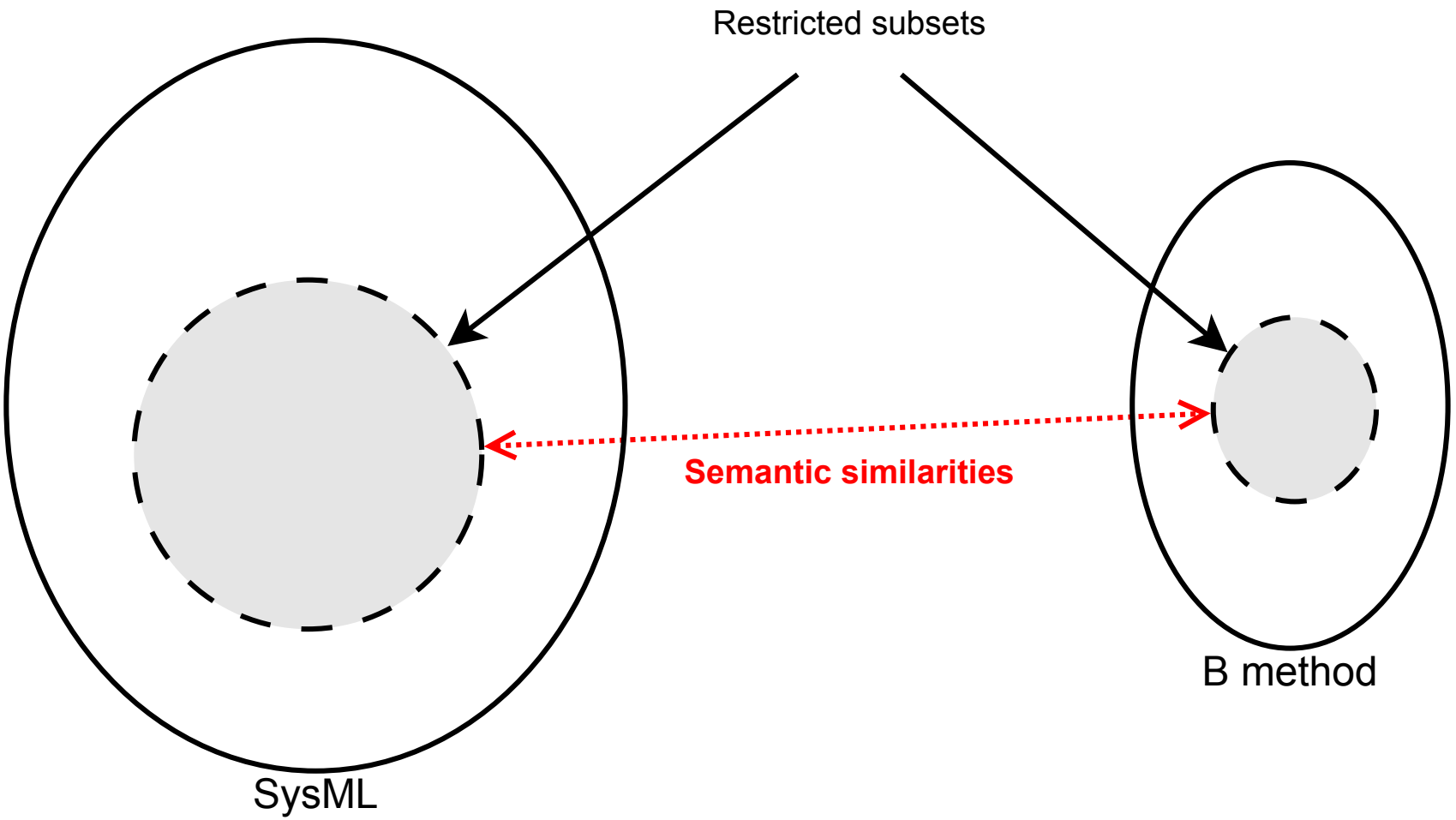
Aligning SysML and B



Aligning SysML and B



Aligning SysML and B



Finding semantic similarities with SysML

Semantic similarities

Features of both languages that are close semantically

- Searching for semantic similarities = **reading official specifications** of both languages and highlighting potentially related parts
- Specifications written in natural language: **subjectivity**
- Formal definitions (ex. B execution semantics in the B book) not taken into account: we look at the **roles** of the features

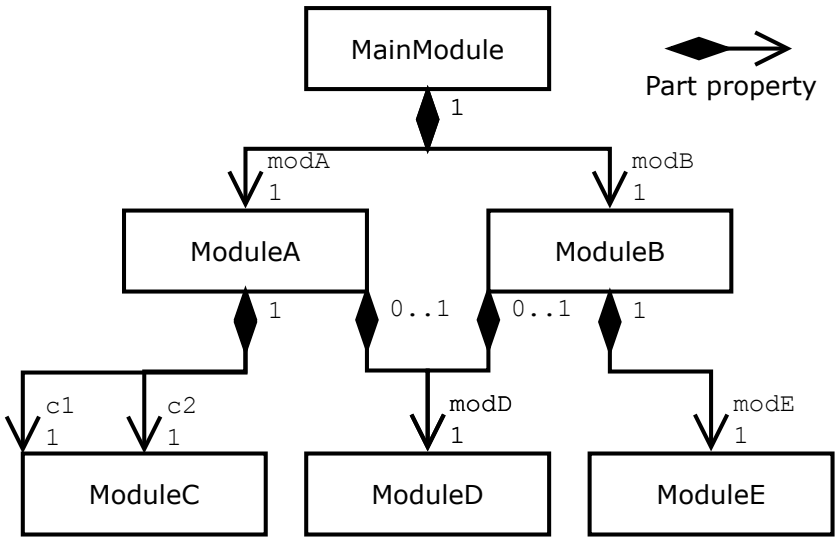
Examples of semantic similarities

B Language Manual	OMG SysML specification
"*A **B module** models a subsystem; it forms a part of a B project.*"	"*A **Block** is a modular unit that describes the structure of a system or element.* "
"***Import** is used to structure a B project into layers, since the implementation of a module is implemented by importing other modules.*"	"*SysML blocks [...] provide the ability to represent a system hierarchy, in which a system at one level is composed of systems at a more basic level. [...] A **part property** holds instances that belong to a larger whole.*"

12 semantic similarities identified in this first approach

Obtained SysML subset

- System decomposed in **blocks** using **part properties** links
- Blocks data stored in **value properties** (integers, boolean, enumerations)
- Behaviors described in **operations** written in Alf
- Invariants declared in **constraint properties** of blocks using the OCL language



Some additions to our SysML subset

Missing counterparts for essential concepts of B

- Need to differentiate the main block of the system
- No notion of abstract data
- Need to differentiate constraints related to subsystems

→ **Profile** with three stereotypes: «main», «abstract», «gluing»

Need for a practical way to design reactive systems

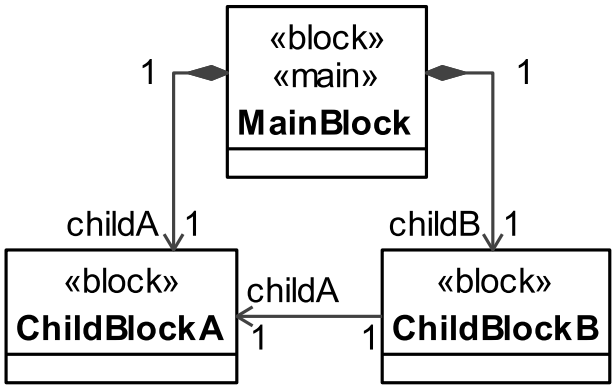
- Existing work of [Sekerenski 1998] on the translation of UML state machines into B

→ Using this work, **state machines** added to our SysML subset

14 rules written in natural language to define our SysML subset

Translating SysML into B

- Mapping directly based on semantic similarities
- 19 transformation rules defined with minimal examples



MACHINE
MainBlock
END

IMPLEMENTATION
MainBlock_i
REFINES
MainBlock
IMPORTS
childA.ChildBlockA,
childB.ChildBlockB
END

MACHINE
ChildBlockA
END

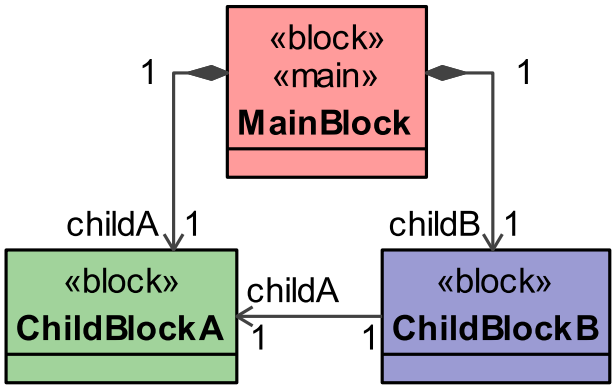
IMPLEMENTATION
ChildBlockA_i
REFINES
ChildBlockA
END

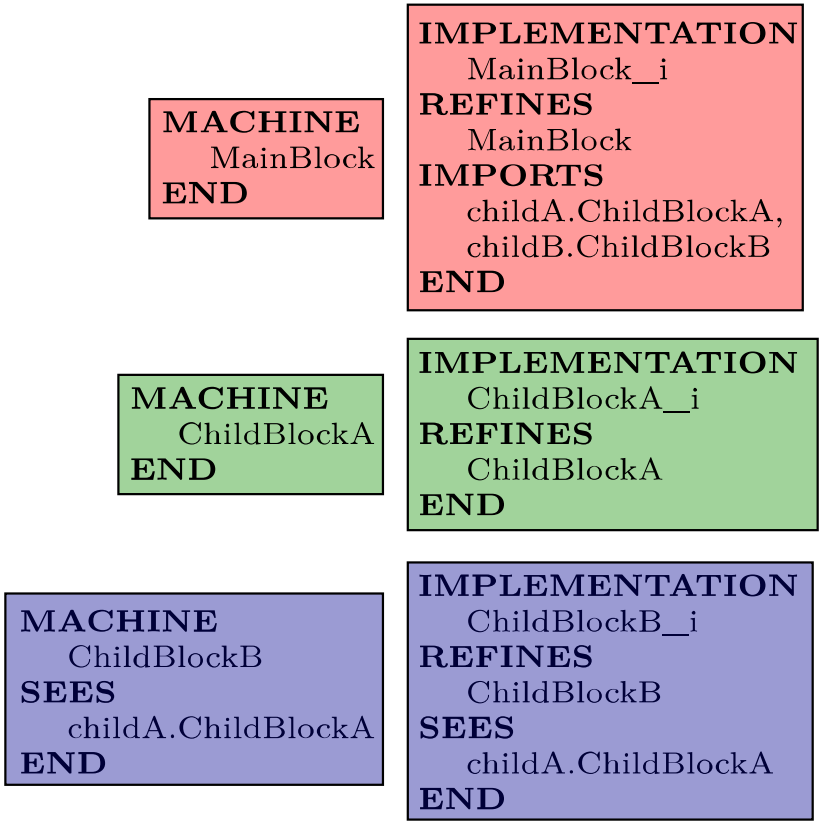
MACHINE
ChildBlockB
SEES
childA.ChildBlockA
END

IMPLEMENTATION
ChildBlockB_i
REFINES
ChildBlockB
SEES
childA.ChildBlockA
END

Translating SysML into B

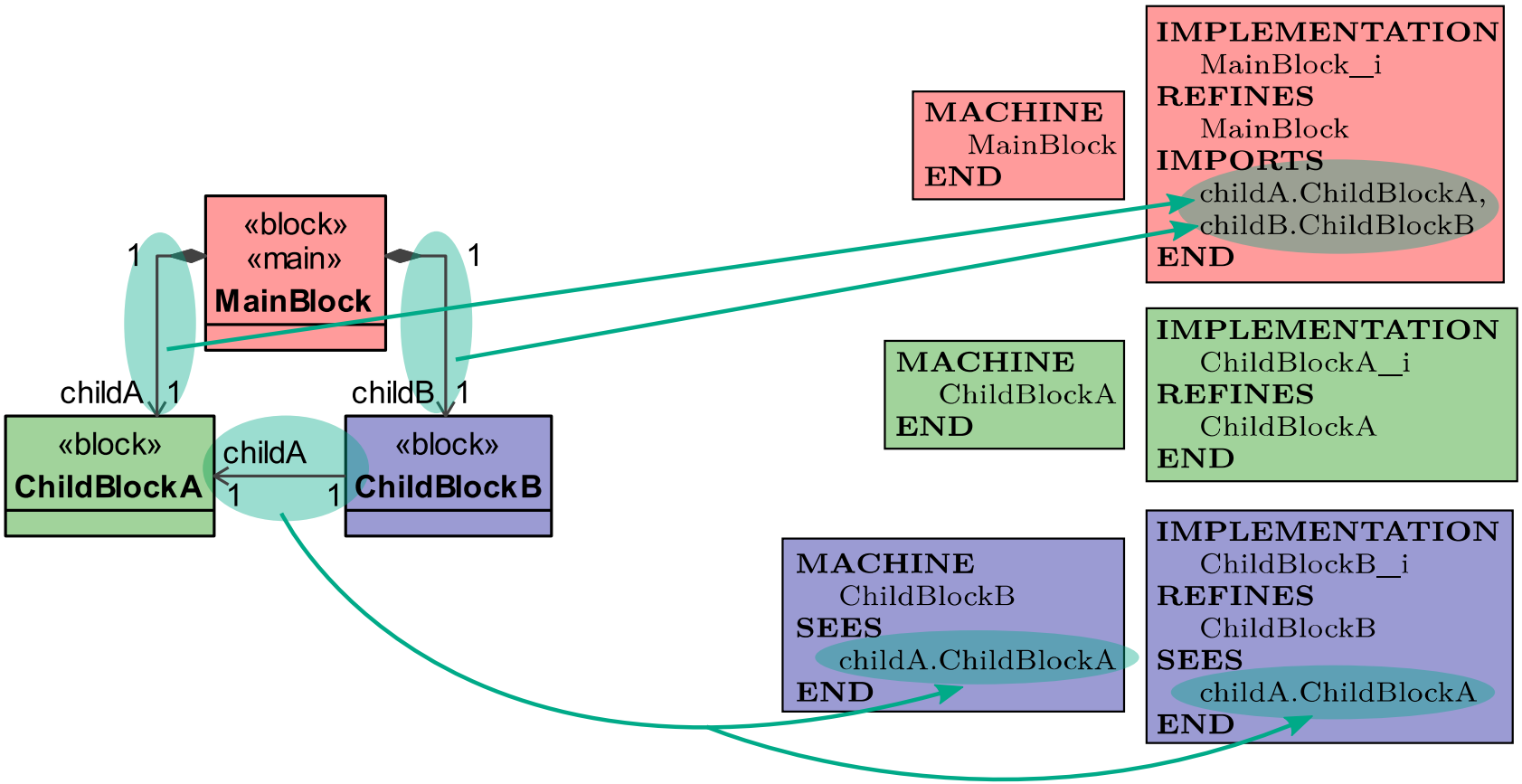
- Mapping directly based on semantic similarities
- 19 transformation rules defined with minimal examples





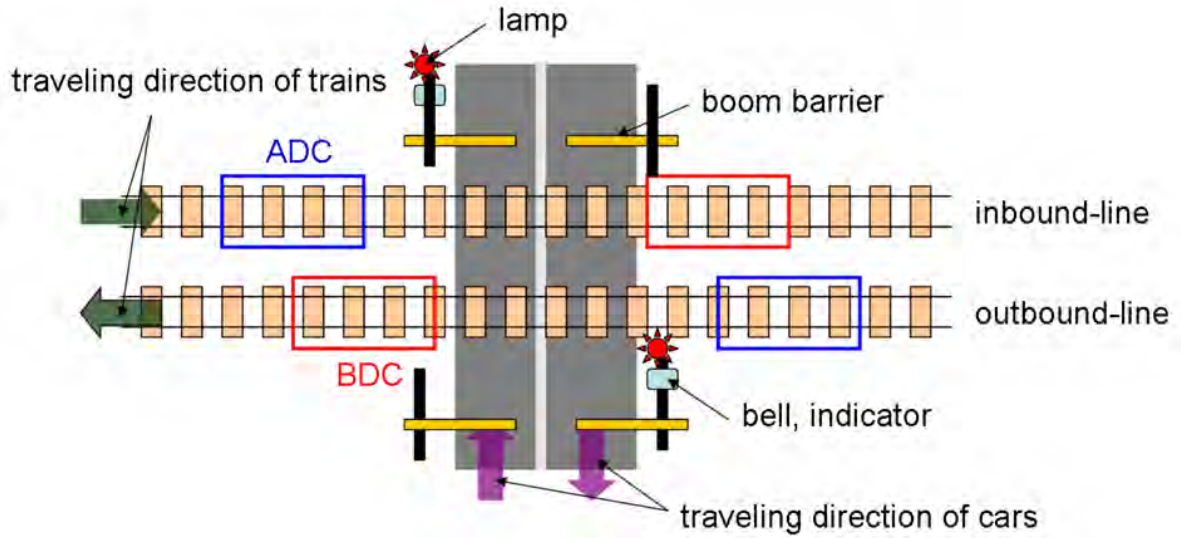
Translating SysML into B

- Mapping directly based on semantic similarities
- 19 transformation rules defined with minimal examples



Case study

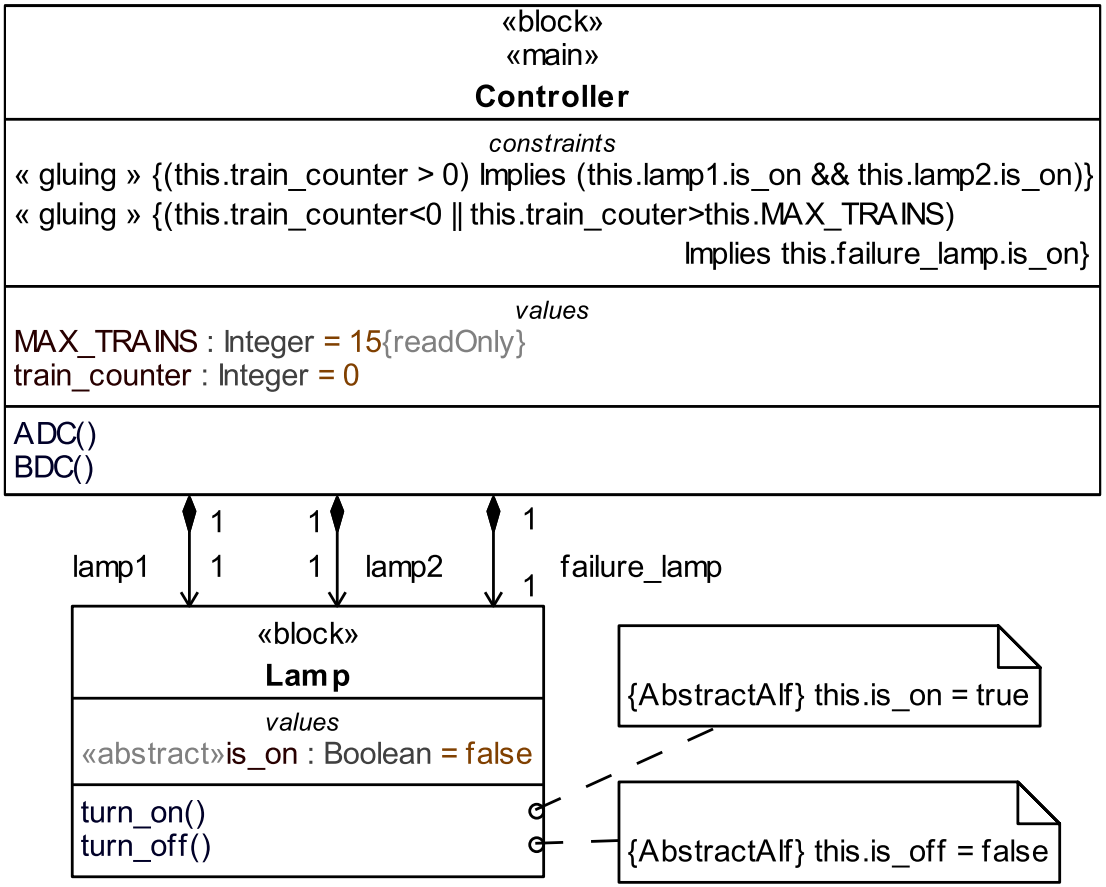
Informal specification of a railway crossing controller



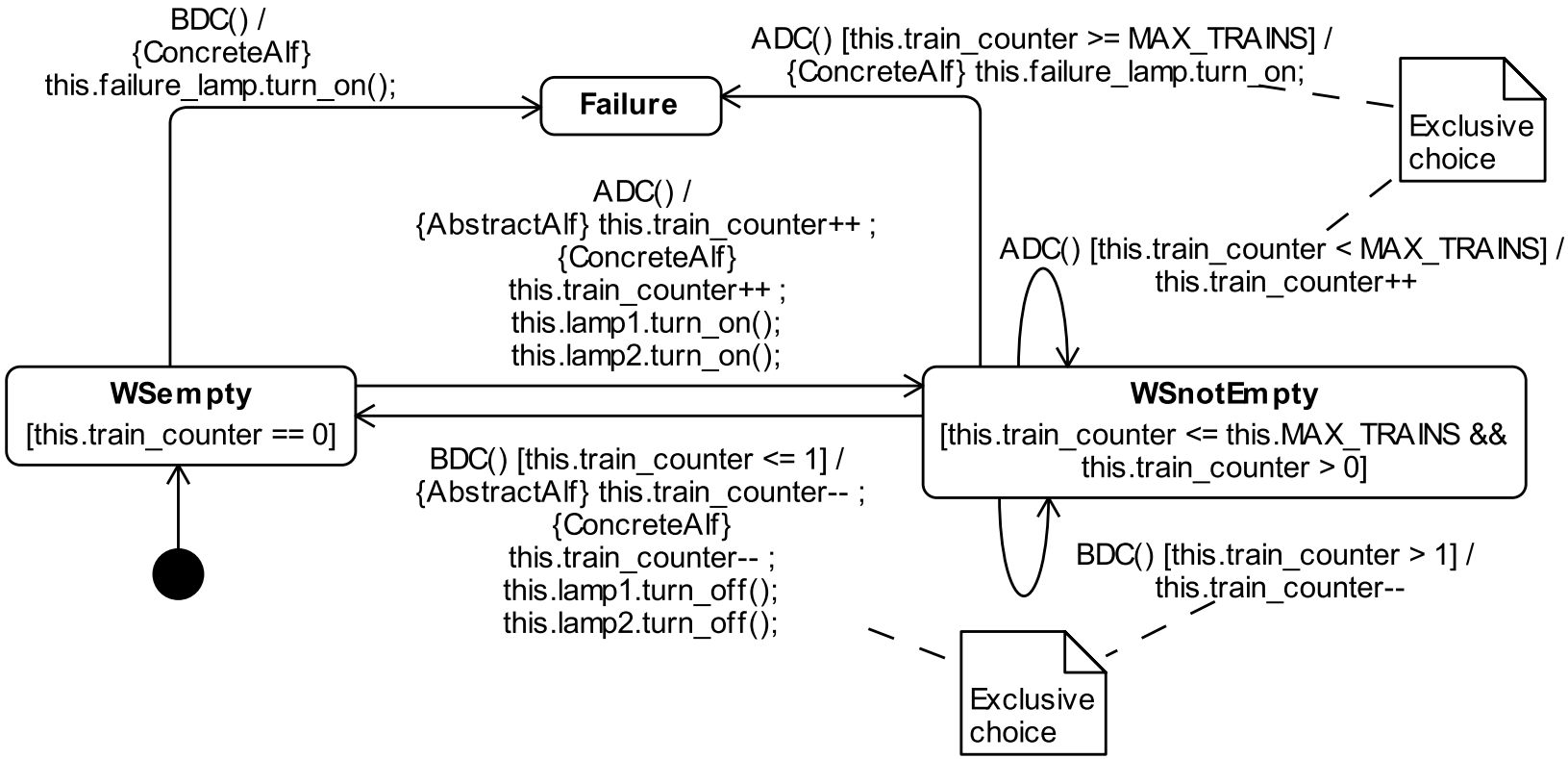
- Two tracks (inbound and outbound) both with a critical section between two train sensors
- Cars may pass on the crossing road
- **Main requirement**: when trains are in the critical sections, barriers, bells, and lamps must be activate accordingly

Case study: SysML block definition diagram

Simplified version: 2 lamps to lit when trains are in the critical sections, 1 lamp to lit when there is a failure



Case study: SysML state machine



Case study: lamp in B

MACHINE Lamp
ABSTRACT_VARIABLES
    is_on
INVARIANT
    is_on $\in \mathbb{B}$
INITIALISATION
    is_on := **FALSE**
OPERATIONS
    turn_on = is_on := **TRUE**;
    turn_off = is_on := **FALSE**
END

IMPLEMENTATION Lamp_i
REFINES Lamp
OPERATIONS
    turn_on = **skip**;
    turn_off = **skip**
END

Case study: controller in B

MACHINE Controller
SETS Controller_states = {WSEmpty,WSnotEmpty,Failure}
CONCRETE_CONSTANTS
MAX_TRAINS
PROPERTIES
MAX_TRAINS ∈ INT ∧ MAX_TRAINS = 15
CONCRETE_VARIABLES
Controller_state, train_counter
INVARIANT
Controller_state∈Controller_states ∧ train_counter∈INT∧
(Controller_state = WSnotEmpty
⇒(train_counter≤MAX_TRAINS∧train_counter>0))∧
(Controller_state = WSEmpty ⇒ (train_counter = 0))
INITIALISATION
Controller_state := WSEmpty || train_counter := 0
OPERATIONS
ADC = ...
BDC = ...
END

IMPLEMENTATION Controller_i
REFINES Controller
IMPORTS lamp1.Lamp, lamp2.Lamp, failure_lamp.Lamp
INVARIANT
((train_counter > 0)
⇒(lamp1.is_on=**TRUE** ∧ lamp2.is_on=**TRUE**))∧
((train_counter < 0 ∨ train_counter > MAX_TRAINS)
⇒ failure_lamp.is_on=**TRUE**)
VALUES
MAX_TRAINS=15
INITIALISATION
Controller_state := WSEmpty ; train_counter := 0
OPERATIONS
ADC = ...
BDC = ...
END

49 Proof Obligations (PO) are generated, all solved automatically

Conclusion

Suggestion of a V&V approach for SysML using the B method

- **Alignment** of both languages
  1. Restricted B method subset
  2. Restricted SysML subset based on the restricted B subset; some gaps filed with a profile and state machines
  3. Translation of the restricted SysML into the restricted B
- **Successful application on an industrial case study**: safety properties are translated and proved
- **Implemented tool**: using Model Driven Engineering (MDE) principles, model transformation written in Kermeta

Further work: combining different V&V approaches for one DSL; bidirectional transformation (traceability, reflecting identified errors on the original model).