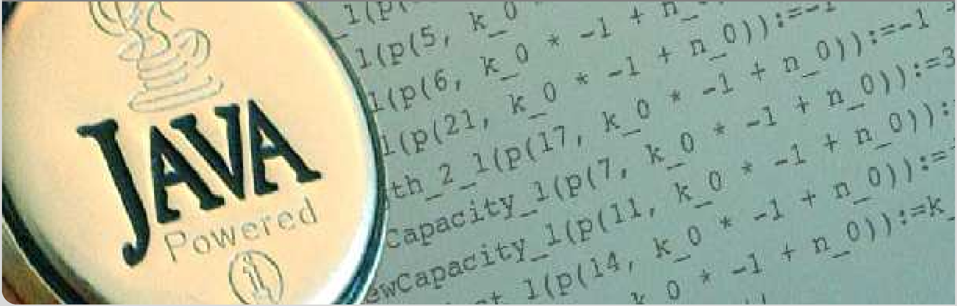
Karlsruhe Institute of Technology

Applications of Formal Verification

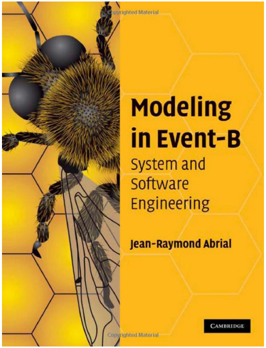
Formal Software Design: Modelling in Event-B

Dr. Vladimir Klebanov · Dr. Mattias Ulbrich | SS 2015

KIT – INSTITUT FÜR THEORETISCHE INFORMATIK



Literatur

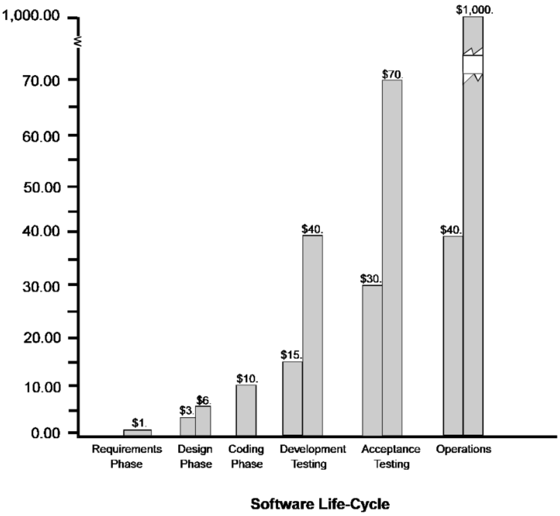


Jean-Raymond Abrial:
Modelling in Event-B
System and Software Enginieering
Cambridge University Press, 2010

Jean-Raymond Abrial:
The B-Book: Assigning programs to meanings
Cambridge University Press, 1996

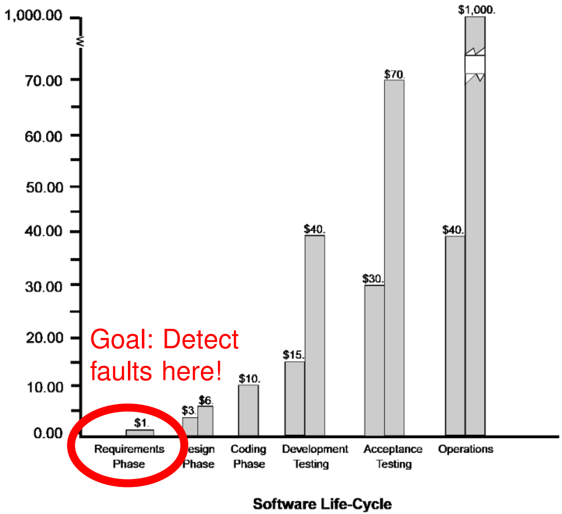
Abstraction and Refinement – Introduction

Late fault recovery is expensive



[“Extra Time Saves Money”, W. Knuffel, Computer Language, 1990]

Late fault recovery is expensive



[“Extra Time Saves Money”, W. Knuffel, Computer Language, 1990]

Reasons for system faults

- Systems are **inherently complex**
- Unconsidered situations, **corner cases**
- **Ambiguous** natural language requirements
- Component **interplay**
- ...

Reasons for system faults

- Systems are **inherently complex**
- Unconsidered situations, **corner cases**
- **Ambiguous** natural language requirements
- Component **interplay**
- ...

Reasons for system faults

- Systems are **inherently complex**
- Unconsidered situations, **corner cases**
- **Ambiguous** natural language requirements
- Component **interplay**
- …

Reasons for system faults

- Systems are **inherently complex**
- Unconsidered situations, **corner cases**
- **Ambiguous** natural language requirements
- Component **interplay**
- ...

Reasons for system faults

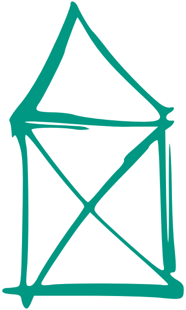
- Systems are **inherently complex**
- Unconsidered situations, **corner cases**
- **Ambiguous** natural language requirements
- Component **interplay**
- ...

Abstraction

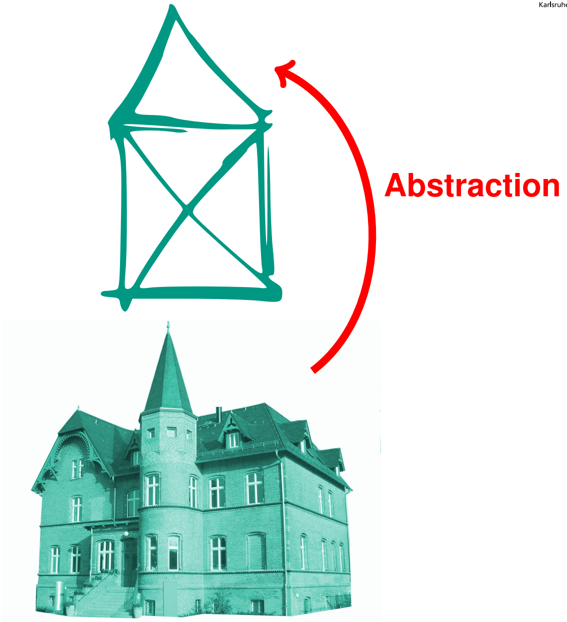
The only tool to **master complexity** is **abstraction**.

CLIFF JONES

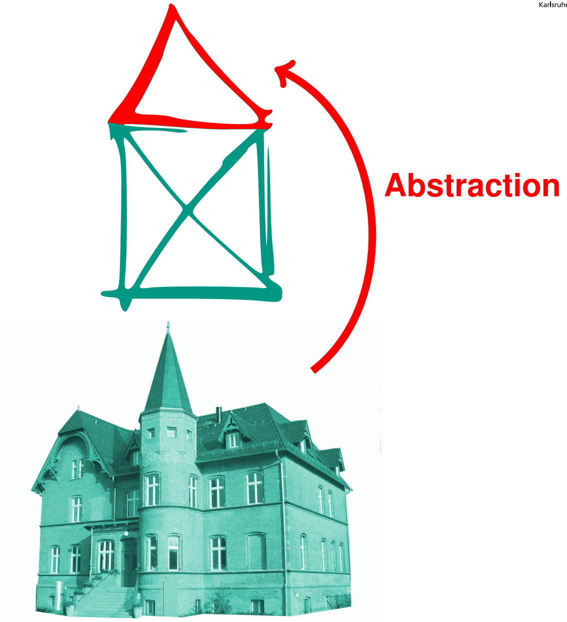
Abstraction and Refinement



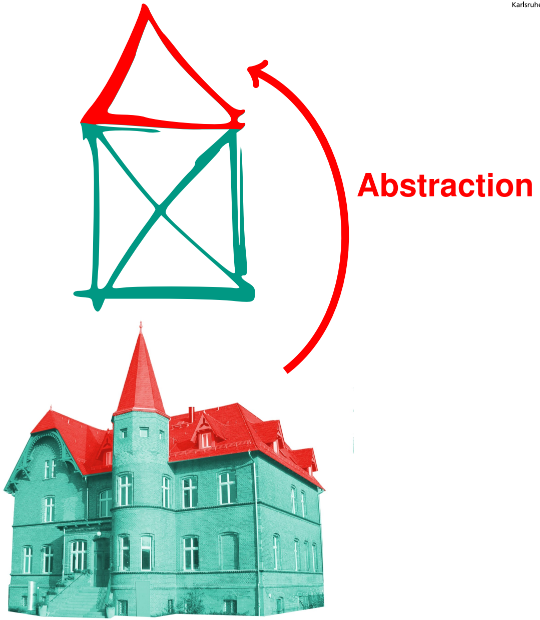
Abstraction and Refinement



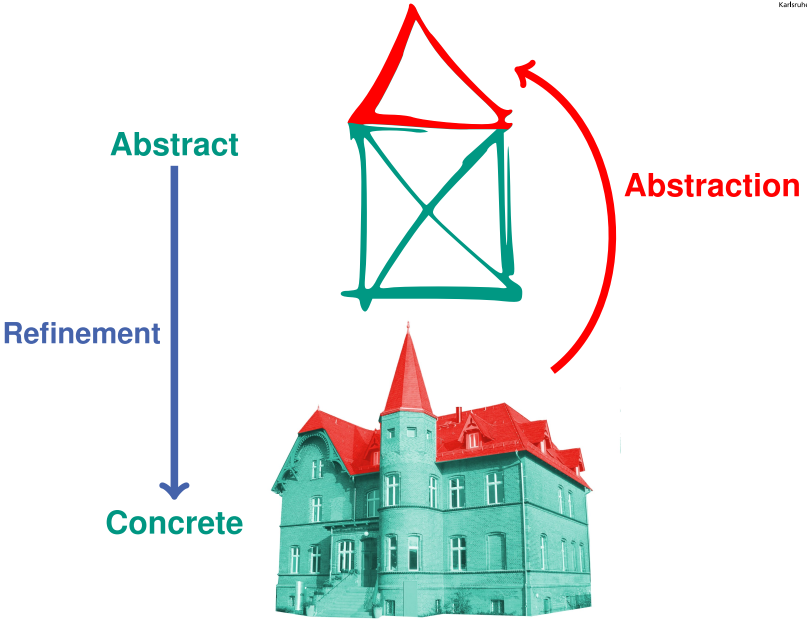
Abstraction and Refinement



Abstraction and Refinement



Abstraction and Refinement



Abstraction

Abstraction

- reduce system complexity
- without removing important properties
- make the model susceptible to formal analysis

and the inverse

Refinement

- enrich abstract model with details
- introduce a new particular aspect
- iterative process: add complexity in a stepwise fashion

Abstraction in Engineering

Abstraction is an important tool in engineering

Established means of abstraction

- Mechanical engineering: BLUEPRINTS
- Electrical engineering: DATASHEETS
- CIRCUIT DIAGRAMS
- Architecture: FLOOR PLANS
- ...

Abstract descriptions remove unnecessary details, concentrate on one aspect

Datasheet – Abstraction

Extracts from datasheet for an IC with four NAND gates

Datasheet – Abstraction

Extracts from datasheet for an IC with four NAND gates

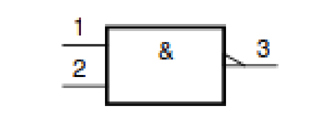
Aspect **Behaviour**

Aspect **Geometry**

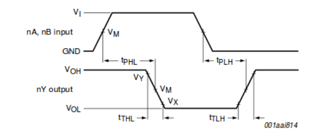
Datasheet – Abstraction

Extracts from datasheet for an IC with four NAND gates

Aspect **Behaviour**



refined to

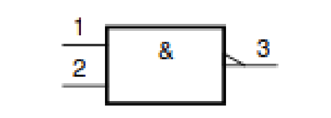


Aspect **Geometry**

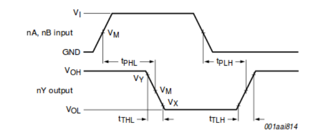
Datasheet – Abstraction

Extracts from datasheet for an IC with four NAND gates

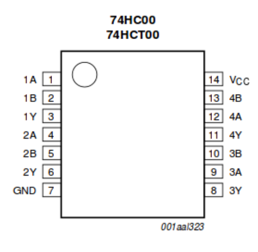
Aspect **Behaviour**



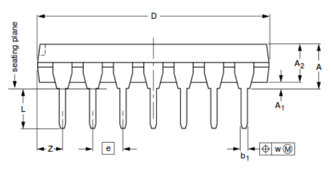
refined to



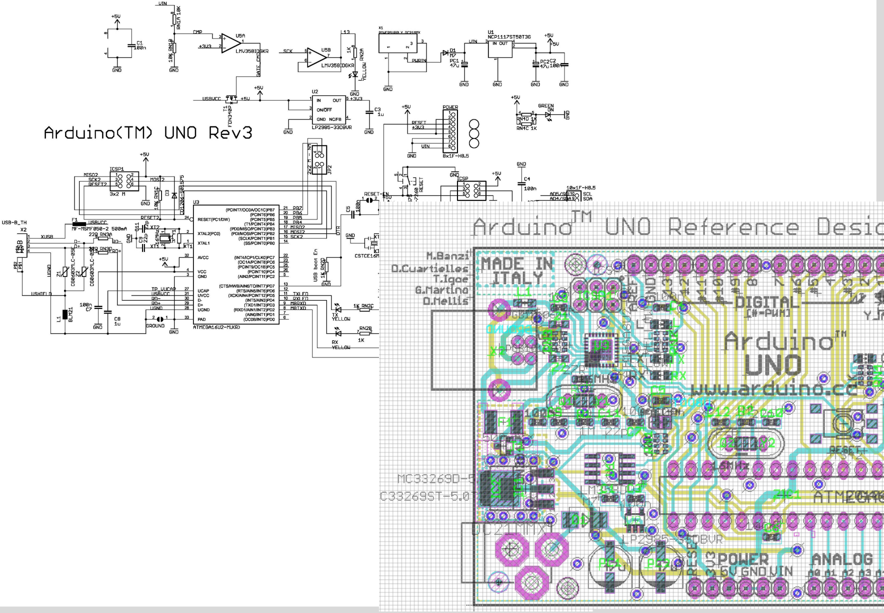
Aspect **Geometry**



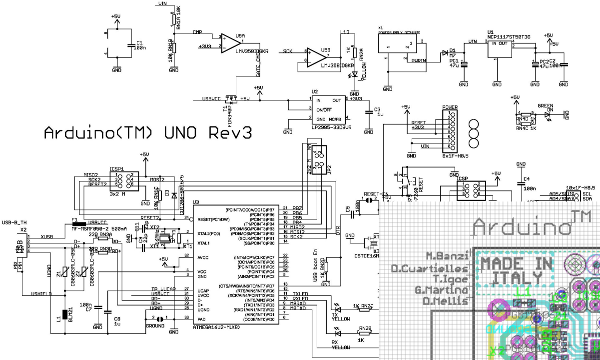
refined to

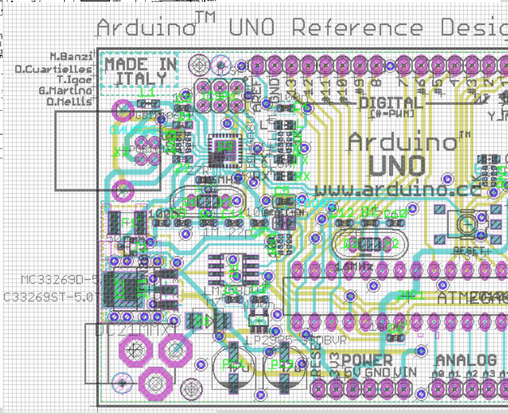


Schematic Diagram vs. PCB Layout



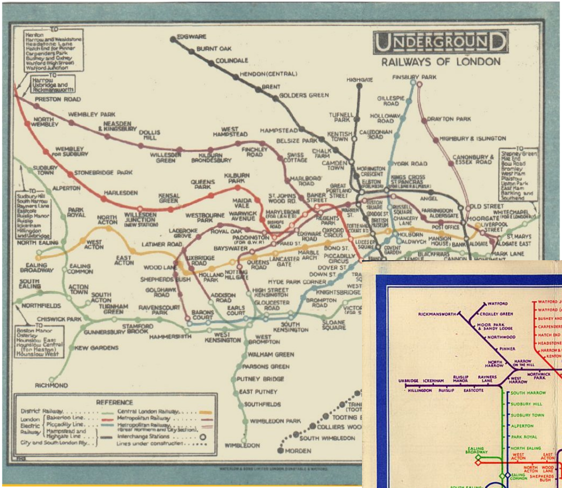
Schematic Diagram vs. PCB Layout



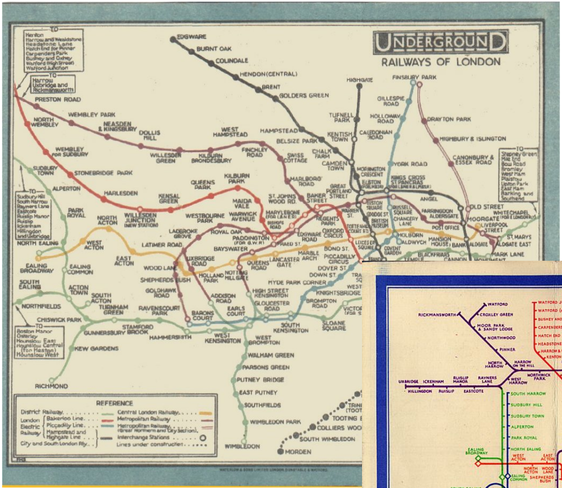


Aspect
"Behaviour"
preserved

Beck diagrams (1931)



Beck diagrams (1931)



Aspect
“Route planning”
is preserved

Property preservation

Abstraction with focus on particular aspect

System properties w.r.t. that aspect must also hold in the abstraction.

Refinement with focus on particular aspect

Properties of abstract model w.r.t. that aspect must be inherited by the refined model.

Examples:

- Abstraction: "The shortest tube travel from Liverpool St. to Westminster has 8 stops and 2 changes."
- Refinement: *Abstract*: Input "$a = 1$" gives output "$b = 1$"
  Concrete: High voltage on pin A gives high voltage on pin B

Property preservation

Abstraction with focus on particular aspect

System properties w.r.t. that aspect must also hold in the abstraction.

Refinement with focus on particular aspect

Properties of abstract model w.r.t. that aspect must be inherited by the refined model.

That's what we will formally prove in the next sections.

Examples:

- Abstraction: "The shortest tube travel from Liverpool St. to Westminster has 8 stops and 2 changes."
- Refinement: *Abstract*: Input "$a = 1$" gives output "$b = 1$"
  Concrete: High voltage on pin A gives high voltage on pin B

"Conceptual" vs "Technical" Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

“Conceptual” vs “Technical” Abstraction

Two areas of abstraction and refinement in formal methods:

Conceptual abstraction

- reduce complexity for more comprehensibility
- focus on a particular system aspect
- provided by designer/developer
- refinement introduces new aspect

That’s what we will look into in the next sections.

Abstraction as a **technique**

- reduce complexity to enhance performance/reach of a tool
- abstract from given predicates to uninterpreted predicates
- computed automatically
- refinement driven by failed proofs
  (Counter-Example Guided Abstraction Refinement, CEGAR)

Event-B – Introduction

Event-B

- EventB is a formalism for modelling and reasoning about discrete systems.
  - for their structure (how can their state be described) and
  - for their behaviour (how can the evolution of their state be described)
- Models are formulated using set theory
- Event-based evolution of the original **B** Method
- Tool-support:
  - RODIN – deductive verification, theorem prover: proofs
  - Pro-B – model checking, animator: counterexamples

Central Concepts

- **Variables and Events**
  - *Variables* model the current state within the state space.
  - *Events* describe operations to model the system behaviour
- **Invariants**
  - properties to be maintained by system
  - formal proof obligations to show that
- **Refinement**
  - Behaviour of refining model is compatible with abstract model
  - formal proof obligation to show that
  - Hence, invariants of abstract model are inherited by concrete model

Central Concepts

- **Variables and Events**
  - *Variables* model the current state within the state space.
  - *Events* describe operations to model the system behaviour
- **Invariants**
  - properties to be maintained by system
  - formal proof obligations to show that
- **Refinement**
  - Behaviour of refining model is compatible with abstract model
  - formal proof obligation to show that
  - Hence, invariants of abstract model are inherited by concrete model

Contexts and Machines

Event-B models

systems state evolution over time, triggered by events

Event-B models consist of **contexts and machines**:

Contexts

Static, rigid, constant parts that *do not* change over time.

Machines

Dynamic, volatile, evolving parts that *do* change over time.

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Contexts and Machines

Event-B models consist of **contexts and machines**:

Contexts

- *Carrier sets* (ground types, universes, “urelements”)
- *Constants* (state-independent symbols, rigid symbols)
- *Axioms* (formulas valid by stipulation)
- *Theorems* (formulas proved valid)

Machines

- *Context references* (which symbols are available)
- *Variables* (state-dependent symbols, non-rigid symbols, program variables)
- *Invariants* (formulas true in every reachable system state)
- *Events* (state transition descriptions)

(Explanations or alternative names in parens)

Introduction by Example

Students and Exams – Requirements

R1 Every **student** must take a final exam in a **subject** of their choice.

R2 They can have **attempts** without yet failing or passing.

R3 Eventually they can **pass** or **fail**, but **never both.**

→ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Introduction by Example

Students and Exams – Requirements

R1 Every **student** must take a final exam in a **subject** of their choice.

R2 They can have **attempts** without yet failing or passing.

R3 Eventually they can **pass** or **fail**, but **never both.**

→ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Introduction by Example

Students and Exams – Requirements

- R1 Every **student** must take a final exam in a **subject** of their choice.
- R2 They can have **attempts** without yet failing or passing.
- R3 Eventually they can **pass** or **fail**, but **never both.**

→ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Introduction by Example

Students and Exams – Requirements

R1 Every **student** must take a final exam in a **subject** of their choice.

R2 They can have **attempts** without yet failing or passing.

R3 Eventually they can **pass** or **fail**, but **never both.**

→ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Introduction by Example

Students and Exams – Requirements

R1 Every **student** must take a final exam in a **subject** of their choice.

R2 They can have **attempts** without yet failing or passing.

R3 Eventually they can **pass** or **fail**, but **never both.**

➜ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Introduction by Example

Students and Exams – Requirements

R1 Every **student** must take a final exam in a **subject** of their choice.

R2 They can have **attempts** without yet failing or passing.

R3 Eventually they can **pass** or **fail**, but **never both**.

→ Identify the **context**, the **state** and the **events** according to the requirements R1–R3.

Exam Context

CONTEXT *ExamCtxt*

Exam Context

CONTEXT *ExamCtxt*

SETS

  STUDENT // see requirement R1

  SUBJECT

Exam Context

CONTEXT *ExamCtxt*

SETS

STUDENT // see requirement R1

SUBJECT

CONSTANTS

maths *physics* *siblings*

Exam Context

CONTEXT *ExamCtxt*

SETS

STUDENT // see requirement R1

SUBJECT

CONSTANTS

maths *physics* *siblings*

AXIOMS

$maths \in SUBJECT$ // type of variables

$physics \in SUBJECT$

Exam Context

CONTEXT *ExamCtxt*

SETS

STUDENT // see requirement R1

SUBJECT

CONSTANTS

maths *physics* *siblings*

AXIOMS

$maths \in SUBJECT$ // type of variables

$physics \in SUBJECT$

$maths \neq physics$ // constants could have same value

Exam Context

CONTEXT *ExamCtxt*

SETS

STUDENT // see requirement R1
SUBJECT

CONSTANTS

maths *physics* *siblings*

AXIOMS

$maths \in SUBJECT$ // type of variables
$physics \in SUBJECT$
$maths \neq physics$ // constants could have same value
$siblings \subseteq STUDENT \times STUDENT$ // function type

Exam Context

CONTEXT *ExamCtxt*

SETS

$STUDENT$ // see requirement R1
$SUBJECT$

CONSTANTS

$maths \quad physics \quad siblings$

AXIOMS

$maths \in SUBJECT$ // type of variables
$physics \in SUBJECT$
$maths \neq physics$ // constants could have same value
$siblings \subseteq STUDENT \times STUDENT$ // function type
$\forall s \cdot s \in STUDENT \Rightarrow (s \mapsto s) \notin siblings$ // irreflexive
// ...

Exam Machine

MACHINE *ExamAbstract*

Exam Machine

MACHINE *ExamAbstract*
SEES *ExamCtxt*

Exam Machine

MACHINE *ExamAbstract*
SEES *ExamCtxt*

VARIABLES
passed *failed*

Exam Machine

MACHINE *ExamAbstract*
SEES *ExamCtxt*

VARIABLES

$passed \quad failed$

INVARIANTS

$passed \subseteq STUDENT \quad failed \subseteq STUDENT$

Exam Machine

MACHINE *ExamAbstract*
SEES *ExamCtxt*

VARIABLES

$passed \quad failed$

INVARIANTS

$passed \subseteq STUDENT \quad failed \subseteq STUDENT$

$passed \cap failed = \varnothing \quad // \text{R3}$

Exam Machine

MACHINE *ExamAbstract*
SEES *ExamCtxt*

VARIABLES

$passed \quad failed$

INVARIANTS

$passed \subseteq STUDENT \qquad failed \subseteq STUDENT$

$passed \cap failed = \varnothing$ // R3

EVENTS

INITIALISATION $\widehat{=}$ ...

ATTEMPTEXAM $\widehat{=}$ ... // R2

PASSEXAM $\widehat{=}$ ... // R3

FAILEXAM $\widehat{=}$ ... // R3

Exam Machine (2)

MACHINE *ExamAbstract*
VARIABLES *passed failed* ...

EVENTS

INITIALISATION $\widehat{=}$

$\quad failed := \varnothing$

$\quad passed := \varnothing$

Exam Machine (2)

MACHINE *ExamAbstract*
VARIABLES *passed* *failed* ...

EVENTS
INITIALISATION $\widehat{=}$
$failed := \varnothing$
$passed := \varnothing$

PASSEXAM $\widehat{=}$
ANY *s* *grade*
WHERE $s \in STUDENT \wedge grade \leq 4$
THEN $passed := passed \cup \{s\}$

Exam Machine (2)

MACHINE *ExamAbstract*
VARIABLES *passed failed* ...

EVENTS
INITIALISATION $\widehat{=}$
$\quad \mathit{failed} := \varnothing$
$\quad \mathit{passed} := \varnothing$

PASSEXAM $\widehat{=}$
ANY *s grade*
WHERE $s \in \mathit{STUDENT} \land \mathit{grade} \leq 4$
THEN $\mathit{passed} := \mathit{passed} \cup \{s\}$

FAILEXAM $\widehat{=}$
ANY *s grade*
WHERE $s \in \mathit{STUDENT} \land \mathit{grade} > 4$
THEN $\mathit{failed} := \mathit{failed} \cup \{s\}$

Invariant violated

MACHINE *ExamAbstract*
VARIABLES *passed failed*
INVARIANTS $passed \cap failed = \varnothing$ ...

EVENTS
PASSEXAM $\widehat{=}$
ANY s $grade$
WHERE $s \in STUDENT \wedge grade \leq 4$
THEN $passed := passed \cup \{s\}$

FAILEXAM $\widehat{=}$
ANY s $grade$
WHERE $s \in STUDENT \wedge grade > 4$
THEN $failed := failed \cup \{s\}$

Invariant violated

MACHINE *ExamAbstract*
VARIABLES *passed failed*
INVARIANTS $\mathit{passed} \cap \mathit{failed} = \varnothing \quad \ldots$

EVENTS
PASSEXAM $\widehat{=}$
  ANY *s grade*
  WHERE $s \in \mathit{STUDENT} \setminus (\mathit{failed} \cup \mathit{passed}) \land \mathit{grade} \leq 4$
  THEN $\mathit{passed} := \mathit{passed} \cup \{s\}$

FAILEXAM $\widehat{=}$
  ANY *s grade*
  WHERE $s \in \mathit{STUDENT} \setminus (\mathit{failed} \cup \mathit{passed}) \land \mathit{grade} > 4$
  THEN $\mathit{failed} := \mathit{failed} \cup \{s\}$

Underspecified model

EVENTS

PASSEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade \leq 4 \wedge s \in \ldots$

THEN $passed := passed \cup \{s\}$

FAILEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade > 4 \wedge s \in \ldots$

THEN $failed := failed \cup \{s\}$

ATTEMPTEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade \in \mathbb{N} \wedge s \in \ldots$

THEN $skip$

Underspecified model

EVENTS

PASSEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade \leq 4 \wedge s \in \ldots$

THEN $passed := passed \cup \{s\}$

FAILEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade > 4 \wedge s \in \ldots$

THEN $failed := failed \cup \{s\}$

ATTEMPTEXAM $\widehat{=}$

ANY s $grade$ WHERE $grade \in \mathbb{N} \wedge s \in \ldots$

THEN $skip$

Additional requirement

R4 Any student may attempt the exam three times and ultimately fails if the fourth attempt is unsuccessful.

Refinement Exams (1)

MACHINE *RefinedExams* REFINES *ExamsAbstract*

Refinement Exams (1)

MACHINE *RefinedExams* REFINES *ExamsAbstract*
VARIABLES *passed attempts*

Refinement Exams (1)

MACHINE *RefinedExams* REFINES *ExamsAbstract*
VARIABLES *passed attempts*
INVARIANTS

$\mathit{attempts} \in \mathit{STUDENT} \to \mathbb{N}$ // typing for *attempts*

$\mathit{failed} = \{s \cdot \mathit{attempts}(s) = 4\}$ // coupling invariant

Refinement Exams (1)

MACHINE *RefinedExams* REFINES *ExamsAbstract*
VARIABLES *passed attempts*
INVARIANTS

$attempts \in STUDENT \rightarrow \mathbb{N}$ // typing for *attempts*

$failed = \{s \cdot attempts(s) = 4\}$ // coupling invariant

EVENTS

INITIALISATION $\widehat{=}$ REFINES INITIALISATION

$passed := \varnothing$

$attempts := \{s \cdot s \in STUDENT \mid (s \mapsto 0)\}$

Refinement Exams (1)

MACHINE *RefinedExams* REFINES *ExamsAbstract*
VARIABLES *passed attempts*
INVARIANTS

$attempts \in STUDENT \to \mathbb{N}$ // typing for *attempts*

$failed = \{s \cdot attempts(s) = 4\}$ // coupling invariant

EVENTS

INITIALISATION $\widehat{=}$ REFINES INITIALISATION

$passed := \varnothing$

$attempts := \{s \cdot s \in STUDENT \mid (s \mapsto 0)\}$

EXAMULTIMATEFAIL $\widehat{=}$ REFINES EXAMFAIL ...
EXAMMISSED $\widehat{=}$ REFINES EXAMATTEMPT ...
EXAMPASSED $\widehat{=}$ REFINES EXAMPASSED ...

Refinement Exams (2)

...

EVENTS

EXAMULTIMATEFAIL $\widehat{=}$ REFINES EXAMFAIL

ANY *s grade*

WHERE ... $\wedge$ $grade > 4 \wedge attempts(s) = 3$

THEN

$attempts(s) := attempts(s) + 1$

EXAMMISSED $\widehat{=}$ REFINES EXAMATTEMPT

ANY *s grade*

WHERE ... $\wedge$ $grade > 4 \wedge attempts(s) < 3$

THEN

$attempts(s) := attempts(s) + 1$

...

Refinement Exams (3)

This refinment takes now also R4 into account.

Refinement preserves invariants

- ! Every possible event of *RefinedExams* is a possible event in *ExamsAbstract*
- $\Rightarrow$ Every invariant of *ExamsAbstract* is also an invariant of *RefinedExams*

We will come back to this more formally ...

Set Theory –
Equipment for formal modelling

Set theory – a universal modelling language

Not only used in Event-B.

Set theory also used for modelling in ...

- Z
- Object-Z, Z++
- (classical) B
- Event-B
- Alloy
- ...

Set Theory

Formal language in Event-B models

Typed First Order Set Theory with Additional Theories

Every term in Event-B has a unqiue type.

Types are *part of the syntax* of Event-B and some expressions are syntactically forbidden:

$$maths \in failed \quad \text{is syntactially invalid.}$$

(remember: $math \in SUBJECT$, $failed \subseteq STUDENT$)

“You can’t compare apples and oranges.”

Set Theory

Formal language in Event-B models

Typed **First Order Set Theory** with Additional Theories

- sets are objects in the logic
- first order axioms define the semantics of sets
- quantification over sets is allowed
- quantification over predicates, functions is not allowed
- (Foundation is a typed Zermelo-Fraenkel axiomatisation)

Set Theory

Formal language in Event-B models

Typed **First Order Set Theory** with Additional Theories

- sets are objects in the logic
- first order axioms define the semantics of sets
- quantification over sets is allowed
- quantification over predicates, functions is not allowed
- (Foundation is a typed Zermelo-Fraenkel axiomatisation)

Set Theory

Formal language in Event-B models

Typed **First Order Set Theory** with Additional Theories

- sets are objects in the logic
- first order axioms define the semantics of sets
- quantification over sets is allowed
- quantification over predicates, functions is not allowed
- (Foundation is a typed Zermelo-Fraenkel axiomatisation)

Set Theory

Formal language in Event-B models

Typed **First Order Set Theory** with Additional Theories

- sets are objects in the logic
- first order axioms define the semantics of sets
- quantification over sets is allowed
- quantification over predicates, functions is not allowed
- (Foundation is a typed Zermelo-Fraenkel axiomatisation)

Set Theory

Formal language in Event-B models

Typed **First Order Set Theory** with Additional Theories

- sets are objects in the logic
- first order axioms define the semantics of sets
- quantification over sets is allowed
- quantification over predicates, functions is not allowed
- (Foundation is a typed Zermelo-Fraenkel axiomatisation)

Set Theory

Formal language in Event-B models

Typed First Order Set Theory with **Additional Theories**

There are additional theories with fixed semantics

- integers
- more theories (datatypes) can be added by user
  (an extension to the system)

Types

1. BOOL and $\mathbb{Z}$ are types
2. Every carrier set declared in a CONTEXT is a type.
3. If T is a type then $\mathbb{P}(T)$ is a type.
   Semantics: $\mathbb{P}(T)$ is the set of all subsets of T (powerset).
4. If T_1, T_2 are types then $T_1 \times T_2$ is a type.
   Semantics: $T_1 \times T_2$ is the set of all ordered pairs (a, b) with $a \in T_1$ and $b \in T_2$ (Cartesian produt).

Every expression E has a unqiue type $\tau(E)$.

Types

1. BOOL and $\mathbb{Z}$ are types
2. Every carrier set declared in a CONTEXT is a type.
3. If T is a type then $\mathbb{P}(T)$ is a type.
   Semantics: $\mathbb{P}(T)$ is the set of all subsets of T (powerset).
4. If T_1, T_2 are types then $T_1 \times T_2$ is a type.
   Semantics: $T_1 \times T_2$ is the set of all ordered pairs (a, b) with $a \in T_1$ and $b \in T_2$ (Cartesian produt).

Every expression E has a unqiue type $\tau(E)$.

Types

1. BOOL and $\mathbb{Z}$ are types
2. Every carrier set declared in a CONTEXT is a type.
3. If T is a type then $\mathbb{P}(T)$ is a type.
   Semantics: $\mathbb{P}(T)$ is the set of all subsets of T (powerset).
4. If T_1, T_2 are types then $T_1 \times T_2$ is a type.
   Semantics: $T_1 \times T_2$ is the set of all ordered pairs (a, b) with $a \in T_1$ and $b \in T_2$ (Cartesian produt).

Every expression E has a unqiue type $\tau(E)$.

Types

1. BOOL and $\mathbb{Z}$ are types
2. Every carrier set declared in a CONTEXT is a type.
3. If T is a type then $\mathbb{P}(T)$ is a type.
   Semantics: $\mathbb{P}(T)$ is the set of all subsets of T (powerset).
4. If T_1, T_2 are types then $T_1 \times T_2$ is a type.
   Semantics: $T_1 \times T_2$ is the set of all ordered pairs (a, b) with $a \in T_1$ and $b \in T_2$ (Cartesian produt).

Every expression E has a unqiue type $\tau(E)$.

Types

1. BOOL and $\mathbb{Z}$ are types
2. Every carrier set declared in a CONTEXT is a type.
3. If T is a type then $\mathbb{P}(T)$ is a type.
   Semantics: $\mathbb{P}(T)$ is the set of all subsets of T (powerset).
4. If T_1, T_2 are types then $T_1 \times T_2$ is a type.
   Semantics: $T_1 \times T_2$ is the set of all ordered pairs (a, b) with $a \in T_1$ and $b \in T_2$ (Cartesian produt).

Every expression E has a unqiue type $\tau(E)$.

Types (2)

Set theory needs not be typed: Everything can be viewed a set.

Reasons to introduce types:

- some specification errors may be detected as syntax errors (even before the verification has started)
- avoid Russell's paradox

Types (2)

Set theory needs not be typed: Everything can be viewed a set.

Reasons to introduce types:

- some specification errors may be detected as syntax errors (even before the verification has started)
- avoid Russell's paradox

Russell's paradox

Assume that the expression $\{s \mid \phi\}$ for any formula ϕ denotes a set. Let $R := \{s \mid s \notin s\}$.

Types (2)

Set theory needs not be typed: Everything can be viewed a set.

Reasons to introduce types:

- some specification errors may be detected as syntax errors (even before the verification has started)
- avoid Russell's paradox

Russell's paradox

Assume that the expression $\{s \mid \phi\}$ for any formula ϕ denotes a set. Let $R := \{s \mid s \notin s\}$.
One observes: $R \in R \iff R \notin R \quad \lightning$

Types (2)

Set theory needs not be typed: Everything can be viewed a set.

Reasons to introduce types:

- some specification errors may be detected as syntax errors (even before the verification has started)
- avoid Russell's paradox

Russell's paradox

Assume that the expression $\{s \mid \phi\}$ for any formula ϕ denotes a set. Let $R := \{s \mid s \notin s\}$. Not allowed with types.
One observes: $R \in R \iff R \notin R \quad ↯$
(This crushed naive set theory in early 1900s.)

Sets

Constructors for sets:

- empty set $\emptyset : \mathbb{P}(S)$

Sets

Constructors for sets:

- empty set $\varnothing : \mathbb{P}(S)$
- set extension $\{\ldots\} : S^* \to \mathbb{P}(S)$
  example: $\{1, 2\} : \mathbb{P}(\mathbb{Z})$

Sets

Constructors for sets:

- empty set $\varnothing : \mathbb{P}(S)$
- set extension $\{\ \ldots\ \} : S^* \to \mathbb{P}(S)$
  example: $\{1, 2\} : \mathbb{P}(\mathbb{Z})$
- carrier sets $C : \mathbb{P}(C)$
  example: $\mathit{STUDENT} : \mathbb{P}(\mathit{STUDENT})$

Sets

Constructors for sets:

- empty set $\varnothing : \mathbb{P}(S)$
- set extension $\{\ \ldots\ \} : S^* \to \mathbb{P}(S)$
  example: $\{1, 2\} : \mathbb{P}(\mathbb{Z})$
- carrier sets $C : \mathbb{P}(C)$
  example: $STUDENT : \mathbb{P}(STUDENT)$
- powerset $\mathbb{P}(\cdot) : \mathbb{P}(S) \to \mathbb{P}(\mathbb{P}(S))$
  example: $\mathbb{P}(\{1, 2\}) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\} : \mathbb{P}(\mathbb{Z})$

Sets

Constructors for sets:

- empty set $\varnothing : \mathbb{P}(S)$
- set extension $\{\ldots\} : S^* \to \mathbb{P}(S)$
  example: $\{1, 2\} : \mathbb{P}(\mathbb{Z})$
- carrier sets $C : \mathbb{P}(C)$
  example: $STUDENT : \mathbb{P}(STUDENT)$
- powerset $\mathbb{P}(\cdot) : \mathbb{P}(S) \to \mathbb{P}(\mathbb{P}(S))$
  example: $\mathbb{P}(\{1, 2\}) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\} : \mathbb{P}(\mathbb{Z})$
- product $\cdot \times \cdot : \mathbb{P}(S) \times \mathbb{P}(T) \to \mathbb{P}(S \times T)$
  example: $BOOL \times \{1\} = \{\{true, 1\}, \{false, 1\}\} : \mathbb{P}(BOOL \times \mathbb{Z})$

Sets

Constructors for sets:

- empty set $\varnothing : \mathbb{P}(S)$
- set extension $\{\ \ldots\ \} : S^* \to \mathbb{P}(S)$
  example: $\{1, 2\} : \mathbb{P}(\mathbb{Z})$
- carrier sets $C : \mathbb{P}(C)$
  example: $STUDENT : \mathbb{P}(STUDENT)$
- powerset $\mathbb{P}(\cdot) : \mathbb{P}(S) \to \mathbb{P}(\mathbb{P}(S))$
  example: $\mathbb{P}(\{1, 2\}) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\} : \mathbb{P}(\mathbb{Z})$
- product $\cdot \times \cdot : \mathbb{P}(S) \times \mathbb{P}(T) \to \mathbb{P}(S \times T)$
  example: $BOOL \times \{1\} = \{\{true, 1\}, \{false, 1\}\} : \mathbb{P}(BOOL \times \mathbb{Z})$
- set comprehension $\{x \cdot \varphi \mid e\}$
  formula φ, term $e : T$, result of type $\mathbb{P}(T)$
  example: $\{x \cdot x \geq 2 \mid x * x\} = \{4, 9, 16, \ldots\}$

Relations

Karlsruhe Institute of Technology

- Relations are sets of pairs (tuples).

Relations

- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$

Relations

- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$
- Pairs $(E_1 \mapsto E_2) : \tau(E_1) \times \tau(E_2)$

Relations

- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$
- Pairs $(E_1 \mapsto E_2) : \tau(E_1) \times \tau(E_2)$
- Domain of a relation *dom*(R)
  $\text{dom}(R) = \{x, y \cdot (x \mapsto y) \in R \mid x\}$
  example: $\text{dom}(E_1 \times E_2) = E_1$

Relations

- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$
- Pairs $(E_1 \mapsto E_2) : \tau(E_1) \times \tau(E_2)$
- Domain of a relation $dom(R)$
  $\mathrm{dom}(R) = \{x, y \cdot (x \mapsto y) \in R \mid x\}$
  example: $\mathrm{dom}(E_1 \times E_2) = E_1$
- Range of a relation $ran(R)$
  $\mathrm{ran}(R) = \{x, y \cdot (x \mapsto y) \in R \mid y\}$
  example: $\mathrm{ran}(E_1 \times E_2) = E_2$

Relations

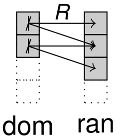
- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$
- Pairs $(E_1 \mapsto E_2) : \tau(E_1) \times \tau(E_2)$
- Domain of a relation *dom*(R)
  dom(R) $= \{x, y \cdot (x \mapsto y) \in R \mid x\}$
  example: dom($E_1 \times E_2$) $= E_1$
- Range of a relation *ran*(R)
  ran(R) $= \{x, y \cdot (x \mapsto y) \in R \mid y\}$
  example: ran($E_1 \times E_2$) $= E_2$
- can be nested: $(E_1 \leftrightarrow E_2) \leftrightarrow E_3$ for a ternary relation *etc.*

Relations

- Relations are sets of pairs (tuples).
- All relations: $E_1 \leftrightarrow E_2 := \mathbb{P}(E_1 \times E_2)$
- Pairs $(E_1 \mapsto E_2) : \tau(E_1) \times \tau(E_2)$
- Domain of a relation *dom*(R)
  $\mathrm{dom}(R) = \{x, y \cdot (x \mapsto y) \in R \mid x\}$
  example: $\mathrm{dom}(E_1 \times E_2) = E_1$ if $E_2 \neq \varnothing$
- Range of a relation *ran*(R)
  $\mathrm{ran}(R) = \{x, y \cdot (x \mapsto y) \in R \mid y\}$
  example: $\mathrm{ran}(E_1 \times E_2) = E_2$ if $E_1 \neq \varnothing$
- can be nested: $(E_1 \leftrightarrow E_2) \leftrightarrow E_3$ for a ternary relation *etc.*

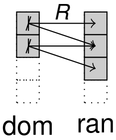
Kinds of relations

- All relations $E_1 \leftrightarrow E_2$

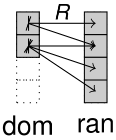


Kinds of relations

- All relations $E_1 \leftrightarrow E_2$

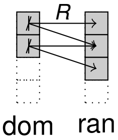


- All surjections $E_1 \leftrightarrow\!\!\!\!\twoheadrightarrow E_2$ $\qquad (\mathrm{ran}(R) = E_2)$

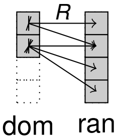


Kinds of relations

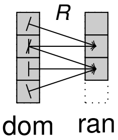
- All relations $E_1 \leftrightarrow E_2$



- All surjections $E_1 \mathrel{\leftrightarrow\!\!\!\!\!\!\twoheadrightarrow} E_2$ $\quad(\text{ran}(R) = E_2)$

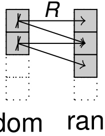


- All total relations $E_1 \mathrel{\twoheadleftarrow\!\!\!\!\!\!\rightarrow} E_2$ $\quad(\text{dom}(R) = E_1)$

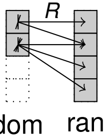


Kinds of relations

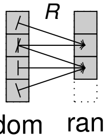
- All relations $E_1 \leftrightarrow E_2$



- All surjections $E_1 \mathrel{\leftarrow\!\!\!\twoheadrightarrow} E_2$ $\quad$ $(\mathrm{ran}(R) = E_2)$



- All total relations $E_1 \mathrel{\twoheadleftarrow\!\!\!\rightarrow} E_2$ $\quad$ $(\mathrm{dom}(R) = E_1)$



- All total surjections $E_1 \mathrel{\twoheadleftarrow\!\!\!\twoheadrightarrow} E_2$

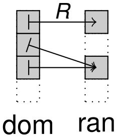
Functional relations

Observation

Every function $f \in A \to B$ can be understood as the relation

$$\{x \cdot x \in A \mid \; x \mapsto f(x) \;\} \quad \in \quad A \leftrightarrow B$$

- Partial functions $E_1 \mathrel{+\!\!\!\to} E_2 \subseteq E_1 \leftrightarrow E_2$
  $(\forall x, y, z \cdot x \mapsto y \in R \land x \mapsto z \in R \Rightarrow y = z)$ $(*)$



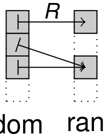
Functional relations

Observation

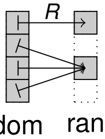
Every function $f \in A \to B$ can be understood as the relation

$$\{x \cdot x \in A \mid x \mapsto f(x)\} \quad \in \quad A \leftrightarrow B$$

- Partial functions $E_1 \mathrel{+\mkern-12mu\rightarrow} E_2 \subseteq E_1 \leftrightarrow E_2$
  $(\forall x, y, z \cdot x \mapsto y \in R \land x \mapsto z \in R \Rightarrow y = z)$ $(*)$



- Total functions $E_1 \to E_2$
  $E_1 \to E_2 \;=\; (E_1 \mathrel{+\mkern-12mu\rightarrow} E_2) \cap (E_1 \mathrel{\leftarrow\mkern-14mu\leftrightarrow} E_2)$
  (both partial function and total relation)



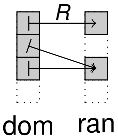
Functional relations

Observation

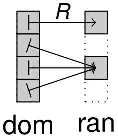
Every function $f \in A \to B$ can be understood as the relation

$$\{x \cdot x \in A \mid\ x \mapsto f(x)\ \} \quad \in \quad A \leftrightarrow B$$

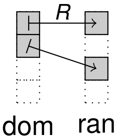
- Partial functions $E_1 \mathrel{+\!\!\!\!\rightarrow} E_2 \subseteq E_1 \leftrightarrow E_2$
  $(\forall x, y, z \cdot x \mapsto y \in R \land x \mapsto z \in R \Rightarrow y = z)\ (*)$



- Total functions $E_1 \to E_2$
  $E_1 \to E_2 \ = \ (E_1 \mathrel{+\!\!\!\!\rightarrow} E_2) \cap (E_1 \mathrel{\leftarrow\!\!\!\!\leftarrow\!\!\!\!\rightarrow} E_2)$
  (both partial function and total relation)



- Injections $E_1 \mathrel{\rightarrowtail\!\!\!\!\!+} E_2$
  $(*) \land (\forall x, y, z \cdot x \mapsto z \in R \land y \mapsto z \in R \Rightarrow x = y)$



Functional relations (2)

Intersection of relation classes give new classes:

- Total injections $E_1 \rightarrowtail E_2 = (E_1 \to E_2) \cap (E_1 ⤔ E_2)$
- Partial surjections $E_1 ⤀ E_2 = (E_1 ⇸ E_2) \cap (E_1 \twoheadleftrightarrow E_2)$
- Total surjections $E_1 \twoheadrightarrow E_2 = (E_1 \to E_2) \cap (E_1 ⤀ E_2)$
- Bijections $E_1 ⤖ E_2 = (E_1 \twoheadrightarrow E_2) \cap (E_1 \rightarrowtail E_2)$

Example: File system

CONTEXT *FileSystemCtx*

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT$, $dirs \subseteq OBJECT$,
$root \in dirs$, $files \cap dirs = \varnothing$

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT$, $dirs \subseteq OBJECT$,
$root \in dirs$, $files \cap dirs = \varnothing$

MACHINE *FileSystem* SEES *FileSystemCtx*
VARIABLES *tree*, *parent*

Example: File system

CONTEXT $\mathit{FileSystemCtx}$
SETS OBJECT
CONSTANTS $\mathit{files}, \mathit{dirs}, \mathit{root}$
AXIOMS $\mathit{files} \subseteq \mathit{OBJECT}, \mathit{dirs} \subseteq \mathit{OBJECT},$
$\mathit{root} \in \mathit{dirs}, \mathit{files} \cap \mathit{dirs} = \varnothing$

MACHINE $\mathit{FileSystem}$ SEES $\mathit{FileSystemCtx}$
VARIABLES $\mathit{tree}, \mathit{parent}$
INVARIANTS
$\mathit{tree} \in \mathit{dirs} \leftrightarrow (\mathit{files} \cup \mathit{dirs})$

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT, dirs \subseteq OBJECT,$
$root \in dirs, files \cap dirs = \varnothing$

MACHINE *FileSystem* SEES *FileSystemCtx*
VARIABLES *tree*, *parent*
INVARIANTS
$tree \in dirs \leftrightarrow (files \cup dirs)$
// most directories (but root) have a parent directory :
$parent \in dirs \pightarrow dirs$

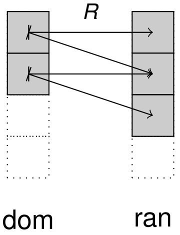
Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT$, $dirs \subseteq OBJECT$,
$root \in dirs$, $files \cap dirs = \varnothing$

MACHINE *FileSystem* SEES *FileSystemCtx*
VARIABLES *tree*, *parent*
INVARIANTS
$tree \in dirs \leftrightarrow (files \cup dirs)$
// most directories (but root) have a parent directory :
$parent \in dirs \nrightarrow dirs$
// more precise
$parent \in (dirs \setminus \{root\}) \rightarrow dirs$

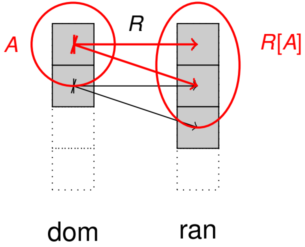
Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \wedge x \in A \mid y\}$



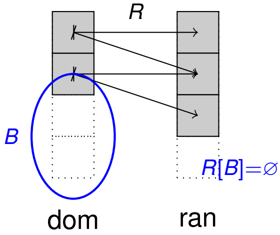
Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \land x \in A \mid y\}$



Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \wedge x \in A \mid y\}$



Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \land x \in A \mid y\}$

- Functional application $\cdot(\cdot) : \mathbb{P}(S \times T) \times S \to T$

$$x = f(e) \iff e \mapsto x \in f \qquad\qquad \{f(e)\} = f[\{e\}]$$

Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \wedge x \in A \mid y\}$

- Functional application $\cdot(\cdot) : \mathbb{P}(S \times T) \times S \to T$

  $$x = f(e) \iff e \mapsto x \in f \qquad\qquad \big\{f(e)\big\} = f\big[\{e\}\big]$$

  Problem: What if $f[\{e\}]$ is not a one-element set?

Relational operations

- Relational application $\cdot[\cdot] : \mathbb{P}(S \times T) \times \mathbb{P}(S) \to \mathbb{P}(T)$
  $R[A] = \{x, y \cdot x \mapsto y \in R \land x \in A \mid y\}$

- Functional application $\cdot(\cdot) : \mathbb{P}(S \times T) \times S \to T$

  $$x = f(e) \iff e \mapsto x \in f \qquad \{f(e)\} = f[\{e\}]$$

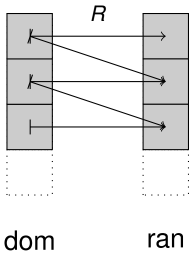
  Problem: What if $f[\{e\}]$ is not a one-element set?
  Solution: Well-definedness needs to be proved
  1. $f \in S \mathrel{+\!\!\!\rightarrow} T$ (not an arbitrary relation in $S \leftrightarrow T$)
  2. $e \in \mathrm{dom}(f)$

  everytime a functional application is used.

Restrictions

Concept

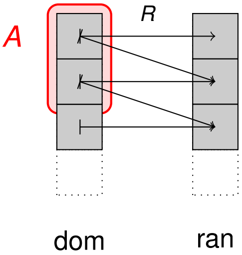
Limit the domain or range of a relation to a subset.



Restrictions

Concept

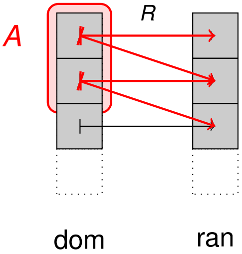
Limit the domain or range of a relation to a subset.



Restrictions

Concept

Limit the domain or range of a relation to a subset.

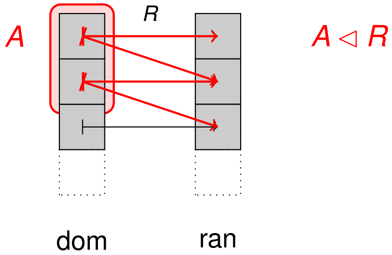


$A \lhd R$

Restrictions

Concept

Limit the domain or range of a relation to a subset.

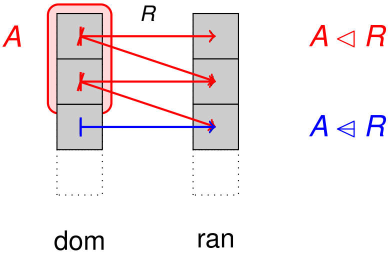


- $A \lhd R \quad := \{x, y \cdot x \mapsto y \in R \land x \in A \mid x \mapsto y\} \subseteq R$
- $A \unlhd R \quad := \{x, y \cdot x \mapsto y \in R \land x \notin A \mid x \mapsto y\} \subseteq R$
- $R \rhd B := \{x, y \cdot x \mapsto y \in R \land y \in B \mid x \mapsto y\} \subseteq R$
- $R \unrhd B := \{x, y \cdot x \mapsto y \in R \land y \notin B \mid x \mapsto y\} \subseteq R$

Restrictions

Concept

Limit the domain or range of a relation to a subset.



- $A \lhd R \quad := \{x, y \cdot x \mapsto y \in R \land x \in A \mid x \mapsto y\} \subseteq R$
- $A \,⩤\, R \quad := \{x, y \cdot x \mapsto y \in R \land x \notin A \mid x \mapsto y\} \subseteq R$
- $R \rhd B := \{x, y \cdot x \mapsto y \in R \land y \in B \mid x \mapsto y\} \subseteq R$
- $R \,⩥\, B := \{x, y \cdot x \mapsto y \in R \land y \notin B \mid x \mapsto y\} \subseteq R$
- *Relational application*: $R[A] = \mathrm{ran}(A \lhd R)$

Override

$$R \mathbin{\lhd\!\!\!+} S := ((\operatorname{dom} S) \mathbin{⩤} R) \cup S$$

$$x \mapsto y \in R \mathbin{\lhd\!\!\!+} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

- "Clear" dom(S) in R and "replace" by S.

Override

$$R \mathbin{⩤} S := ((\operatorname{dom} S) \mathbin{⩤} R) \cup S$$

$$x \mapsto y \in R \mathbin{⩤} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

- "Clear" $\operatorname{dom}(S)$ in R and "replace" by S.
- Special case: $f \in A \to B, g \in A \nrightarrow B$ implies $f \mathbin{⩤} g \in A \to B$

Override

$$R \mathbin{\lhd\!\!\!+} S := ((\operatorname{dom} S) \mathbin{⩤} R) \cup S$$

$$x \mapsto y \in R \mathbin{\lhd\!\!\!+} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

- "Clear" $\operatorname{dom}(S)$ in R and "replace" by S.
- Special case: $f \in A \to B, g \in A \nrightarrow B$ implies $f \mathbin{\lhd\!\!\!+} g \in A \to B$
- $f \mathbin{\lhd\!\!\!+} \{x \mapsto y\}$ updates function f in one place x

Override

$$R \mathrel{\lhd\!\!\!+} S := ((\operatorname{dom} S) \mathbin{\lhd\!\!\!-} R) \cup S$$

$$x \mapsto y \in R \mathrel{\lhd\!\!\!+} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

- "Clear" dom(S) in R and "replace" by S.
- Special case: $f \in A \to B, g \in A \mathrel{+\!\!\!\!\to} B$ implies $f \mathrel{\lhd\!\!\!+} g \in A \to B$
- $f \mathrel{\lhd\!\!\!+} \{x \mapsto y\}$ updates function f in one place x

- Caution: $\mathbin{\lhd\!\!\!-}$ and $\mathrel{\lhd\!\!\!+}$ are different symbols

Override

$$R \mathbin{\lhd\!\!\!+} S := ((\operatorname{dom} S) \lhd R) \cup S$$

$$x \mapsto y \in R \mathbin{\lhd\!\!\!+} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

- “Clear” $\operatorname{dom}(S)$ in R and “replace” by S.
- Special case: $f \in A \to B, g \in A \mathrel{+\!\!\!\rightarrow} B$ implies $f \mathbin{\lhd\!\!\!+} g \in A \to B$
- $f \mathbin{\lhd\!\!\!+} \{x \mapsto y\}$ updates function f in one place x

- Caution: $\lhd$ and $\mathbin{\lhd\!\!\!+}$ are different symbols
- Syntax sometimes $\oplus$ instead of $\mathbin{\lhd\!\!\!+}$

Override

$$R \mathbin{\lessdot\!\!\!+} S := ((\operatorname{dom} S) \lhd R) \cup S$$

$$x \mapsto y \in R \mathbin{\lessdot\!\!\!+} S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \operatorname{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \operatorname{dom}(S) \end{cases}$$

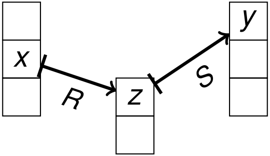
- "Clear" $\operatorname{dom}(S)$ in R and "replace" by S.
- Special case: $f \in A \to B, g \in A \nrightarrow B$ implies $f \mathbin{\lessdot\!\!\!+} g \in A \to B$
- $f \mathbin{\lessdot\!\!\!+} \{x \mapsto y\}$ updates function f in one place x

- Caution: $\lhd$ and $\mathbin{\lessdot\!\!\!+}$ are different symbols
- Syntax sometimes $\oplus$ instead of $\mathbin{\lessdot\!\!\!+}$
- Compare *Updates* in Dynamic Logic for KeY.

Forward composition

$$x \mapsto y \in R\,;S \iff \exists z \cdot x \mapsto z \in R \wedge z \mapsto y \in S$$

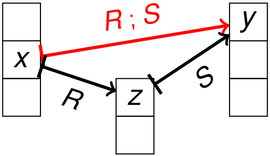
$x \mapsto y$ is in the composition $R\,;S$ if there is a transmitting element z with both $x \mapsto z \in R$ and $z \mapsto y \in S$.



Forward composition

$$x \mapsto y \in R\,;S \iff \exists z \cdot x \mapsto z \in R \land z \mapsto y \in S$$

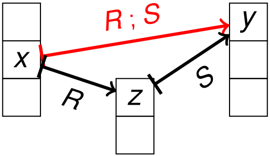
$x \mapsto y$ is in the composition $R\,;S$ if there is a transmitting element z with both $x \mapsto z \in R$ and $z \mapsto y \in S$.



Forward composition

$$x \mapsto y \in R\,;S \iff \exists z \cdot x \mapsto z \in R \land z \mapsto y \in S$$

$x \mapsto y$ is in the composition $R\,;S$ if there is a transmitting element z with both $x \mapsto z \in R$ and $z \mapsto y \in S$.



(There is also backward composition $R \circ S = S\,;R$)

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT$, $dirs \subseteq OBJECT$,
$root \in dirs$, $files \cap dirs = \varnothing$

MACHINE *FileSystem* SEES *FileSystemCtx*
VARIABLES *tree*, *depth*
INVARIANTS
$tree \in dirs \leftrightarrow (files \cup dirs) \quad \wedge \quad depth \in dirs \rightarrow \mathbb{N} \quad \wedge$

Example: File system

CONTEXT *FileSystemCtx*
SETS *OBJECT*
CONSTANTS *files*, *dirs*, *root*
AXIOMS $files \subseteq OBJECT$, $dirs \subseteq OBJECT$,
$root \in dirs$, $files \cap dirs = \varnothing$

MACHINE *FileSystem* SEES *FileSystemCtx*
VARIABLES *tree*, *depth*
INVARIANTS

$$tree \in dirs \leftrightarrow (files \cup dirs) \quad \land \quad depth \in dirs \to \mathbb{N} \quad \land$$
$$\forall d \cdot \big((depth(d) > 0 \Rightarrow depth[tree[\{d\}]] = \{depth(d) - 1\})$$
$$\land\ (depth(d) = 0 \Rightarrow \{d\} \lhd tree \rhd files = \varnothing)\big)$$

Event-B –
Events

Machine (systematic)

MACHINE *name*

SEES *context*

VARIABLES $\overline{vars}$

INVARIANTS $inv(\overline{vars})$

EVENTS

$\ldots$

END

The symbols in context can be used in *inv* even if not mentioned explicitly.

Events

EVENT M

// the following are the parameters,
// the input signals, nondeterministic choices
ANY $\overline{prms}$

// the preconditions, conditions on the input values
WHERE $guard(\overline{vars}, \overline{prms})$

// evolution of the program variables when the event "fires"
THEN
    actions
END

There is one more contruct (WITH) that we omit here.

Actions (Generalised Substitutions)

Deterministic actions

- “Assignment” $x := t$
- Variable x and term t must have same type ($\tau(t) = \tau(x)$)
- After event, x has value of expression t

Actions (Generalised Substitutions)

Deterministic actions

- “Assignment” $x := t$
- Variable x and term t must have same type $(\tau(t) = \tau(x))$
- After event, x has value of expression t

Actions (Generalised Substitutions)

Deterministic actions

- "Assignment" $x := t$
- Variable x and term t must have same type $(\tau(t) = \tau(x))$
- After event, x has value of expression t

Actions (Generalised Substitutions)

Deterministic actions

- "Assignment" $x := t$
- Variable x and term t must have same type ($\tau(t) = \tau(x)$)
- After event, x has value of expression t

Actions (Generalised Substitutions)

Deterministic actions

- "Assignment" $x := t$
- Variable x and term t must have same type ($\tau(t) = \tau(x)$)
- After event, x has value of expression t

Example:

```
THEN
  x := y
  y := x
END    // swaps values of variables x, y.
```

Unmentioned variable z does not change.

Remember the updates in KeY: $\{x := y \| y := x\}$ has same effects.

Actions (Generalised Substitutions)

Nondeterministic actions

$$x :| \varphi \quad \text{means} \quad \text{“choose } x \text{ such that } \varphi\text{”}$$

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Actions (Generalised Substitutions)

Nondeterministic actions

$$x :| \varphi \quad \text{means} \quad \text{“choose } x \text{ such that } \varphi\text{”}$$

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Actions (Generalised Substitutions)

Nondeterministic actions

$$x :| \varphi \quad \text{means} \quad \text{“choose } x \text{ such that } \varphi\text{”}$$

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Actions (Generalised Substitutions)

Nondeterministic actions

$x :\mid \varphi$ means "choose x such that φ"

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Actions (Generalised Substitutions)

Nondeterministic actions

$$x :\mid \varphi \quad \text{means} \quad \text{“choose } x \text{ such that } \varphi\text{”}$$

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Actions (Generalised Substitutions)

Nondeterministic actions

$x :\mid \varphi$ means "choose x such that φ"

- Actions can have more than one resolution
- φ is called the before-after-predicate (BAP)
- variables without tick: before-state
- variables with tick: after-state.

Example:

$$x, y :\mid x' = y' \wedge y' > y$$

After the action x and y are equal and y is strictly greater than before the action.

Actions (Generalised Substitutions)

Normal form

Every action can be defined as a before-after-predicate

$$bap(\overline{vars}, \overline{vars'}, \overline{prms})$$

with

1. $\overline{vars}$ the machines variables before the action
2. $\overline{vars'}$ the machine variables after the action
3. $\overline{prms}$ the parameters of the event

- $x := t$ is short for $x :\mid x' = t$
- $x :\in S$ is short for $x :\mid x' \in S$

Initialisation

- Values of the machine in the beginning?

Initialisation

- Values of the machine in the beginning?
- Initial values defined by the special event INITIALISATION.

Initialisation

- Values of the machine in the beginning?
- Initial values defined by the special event INITIALISATION.
- before-after-predicate bap_{init} and guard grd_{init} must not refer to *vars*,
  there is no "before-state".

Initialisation

Karlsruhe Institute of Technology

- Values of the machine in the beginning?
- Initial values defined by the special event INITIALISATION.
- before-after-predicate bap_{init} and guard grd_{init} must not refer to *vars*,
  there is no "before-state".
- After the first state, only normal events trigger.

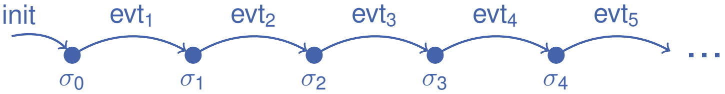
Machine Semantics

Machine variables $\overline{vars} := v_1, ..., v_k$ with types $\overline{T} = T_1 \times ... \times T_k$.

A state $\sigma \in \overline{T}$ is a vector, variable assignment.

A trace is a sequence of states $\sigma_0, \sigma_1, \ldots$ such that

- first state σ_0 is result of the initialisation event
- every state σ_i results from an event which operates on σ_{i-1} (for every $i > 0$).



The semantics of a machine M is the set of all traces possible for M.

Event Parameters

Sources for indeterminism

- indeterministic choices in bap's (cf. $:\in$, $:|$)
- event parameters

Event parameter may model:

- content of messages passed around
- indeterministic user input
- unpredictable environment actions
- a number, amount of data to operate with
- …

Technically event parameters can be removed and replaced by existential quantifiers.

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$
with

- $\exists prms_{init} \cdot grd_{init}(prms_{init}) \wedge bap_{init}(t(0), prms_{init})$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$
with

- $\exists prms_{init} \cdot grd_{init}(prms_{init}) \land bap_{init}(t(0), prms_{init})$
- For $n \in \mathbb{N}_1$, there is $e \in EVENTS$ such that
  $\exists prms_e \cdot grd_e(t(i-1), prms_e) \land bap_e(t(i-1), t(i), prms_e)$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$
with

- $\exists prms_{init} \cdot grd_{init}(prms_{init}) \wedge bap_{init}(t(0), prms_{init})$
- For $n \in \mathbb{N}_1$, there is $e \in EVENTS$ such that $\exists prms_e \cdot grd_e(t(i-1), prms_e) \wedge bap_e(t(i-1), t(i), prms_e)$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$
with

- $\exists prms_{init} \cdot grd_{init}(prms_{init}) \wedge bap_{init}(t(0), prms_{init})$
- For $n \in \mathbb{N}_1$, there is $e \in EVENTS$ such that
  $\exists prms_e \cdot grd_e(t(i-1), prms_e) \wedge bap_e(t(i-1), t(i), prms_e)$

Partial, finite trace trace: $t \in 0..n \to \overline{T}$

Semantics (more formally)

State space: $\overline{T} = T_1 \times \ldots \times T_k$

Trace: $t \in \mathbb{N} \to \overline{T}$
with

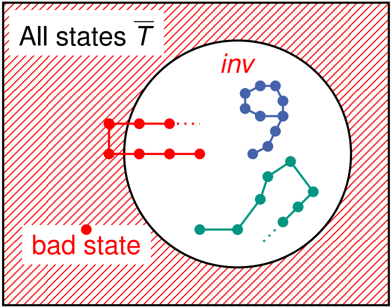
- $\exists prms_{init} \cdot grd_{init}(prms_{init}) \land bap_{init}(t(0), prms_{init})$
- For $n \in \mathbb{N}_1$, there is $e \in EVENTS$ such that
  $\exists prms_e \cdot grd_e(t(i-1), prms_e) \land bap_e(t(i-1), t(i), prms_e)$

Partial, finite trace trace: $t \in 0..n \to \overline{T}$

Deadlock: no event e can be triggered, i.e.
$\forall prms_e \cdot \neg grd_e(t(n), prms_e)$ for all events e.

Invariants

SAFETY: Do all states reachable by *M* satisfy *inv*?



The red trace violates the invariant in two states.

Proof Obligation INV

To show that $inv(\overline{vars})$ is an invariant for machine M,
one proves for every event:

$$
\boxed{
\begin{array}{ll}
 & \text{Invariants} \\
 & \text{Guards of the event} \\
 & \text{Before-after-predicate of the thevent} \\
\Rightarrow & \\
 & \text{modified invariant}
\end{array}
}
$$

Proof Obligation INV

To show that $inv(\overline{vars})$ is an invariant for machine M, one proves:

1. $\forall \overline{prms}, \overline{vars'} \cdot$
   $grd_{init}(\overline{prms}) \land bap_{init}(\overline{vars'}, \overline{prms}) \rightarrow inv(\overline{vars'})$
   (Invariant initally valid)

Proof Obligation INV

To show that $inv(\overline{vars})$ is an invariant for machine M, one proves:

1. $\forall \overline{prms}, \overline{vars'} \cdot$
   $grd_{init}(\overline{prms}) \land bap_{init}(\overline{vars'}, \overline{prms}) \to inv(\overline{vars'})$
   (Invariant initally valid)

2. $\forall \overline{prms}, \overline{vars}, \overline{vars'} \cdot$
   $inv(\overline{vars}) \land grd_e(\overline{vars}, \overline{prms}) \land$
   $bap_e(\overline{vars}, \overline{vars'}, \overline{prms}) \to inv(\overline{vars'})$
   for every event e in M.
   (Events preserve invariant)

Proof Obligation INV

To show that $inv(\overline{vars})$ is an invariant for machine M, one proves:

1. $\forall \overline{prms}, \overline{vars'} \cdot$
   $grd_{init} \wedge bap_{init} \rightarrow inv$
   (Invariant initally valid)

2. $\forall \overline{prms}, \overline{vars}, \overline{vars'} \cdot$
   $inv \wedge grd_e \wedge$
   $bap_e \rightarrow inv'$
   for every event e in M.
   (Events preserve invariant)

Proof Obligation INV

To show that $inv(\overline{vars})$ is an invariant for machine M, one proves:

1. $\forall \overline{prms}, \overline{vars'} \cdot$
   $grd_{init} \wedge bap_{init} \rightarrow inv$
   (Invariant initally valid)

2. $\forall \overline{prms}, \overline{vars}, \overline{vars'} \cdot$
   $inv \wedge grd_e \wedge$
   $bap_e \rightarrow inv'$
   for every event e in M.
   (Events preserve invariant)

Note: Proof Obligation INV is a sufficient criterion, but not necessary. Necessary for *inductive invariants*.

Inductive Invariant

MACHINE *IndInv*
VARIABLES x INVARIANTS $x \in \mathbb{Z}$ $x \geq 0$
EVENTS

INITIALISATION $\widehat{=}$
$x := 2$

STEP $\widehat{=}$
$x := 2 * (x - 1)$

There is only one trace:

$$(2, 2, 2, 2, \ldots)$$

invariant is fulfilled.

Inductive Invariant – Won't prove

Proof obligation INV for event STEP

$$inv(x) \quad \wedge \quad grd(x) \quad \wedge \qquad bap(x, x') \qquad \rightarrow \quad inv(x')$$

Inductive Invariant – Won't prove

Proof obligation INV for event STEP

$$
\begin{array}{ccccccc}
inv(x) & \wedge & grd(x) & \wedge & bap(x, x') & \rightarrow & inv(x') \\
x \geq 0 & & & \wedge & x' = 2 * (x - 1) & \rightarrow & x' \geq 0
\end{array}
$$

Inductive Invariant – Won't prove

Proof obligation INV for event STEP

$$
\begin{array}{ccccccc}
inv(x) & \wedge & grd(x) & \wedge & bap(x, x') & \rightarrow & inv(x') \\
x \geq 0 & & & \wedge & x' = 2 * (x - 1) & \rightarrow & x' \geq 0
\end{array}
$$

$\lightning$ This is not valid! Invariant is not inductive. $\lightning$

Counter-example: $x = 0, x' = -2$

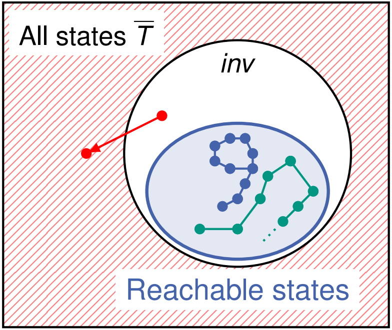
Inductive Invariant – Won't prove

Proof obligation INV for event STEP

$$\begin{array}{ccccccc} inv(x) & \wedge & grd(x) & \wedge & bap(x, x') & \rightarrow & inv(x') \\ x \geq 0 & & & \wedge & x' = 2 * (x - 1) & \rightarrow & x' \geq 0 \end{array}$$

↯ This is not valid! Invariant is not inductive. ↯

Counter-example: $x = 0, x' = -2$



Feasibility Proof Obligation FIS

Show that every action is feasible if the guard is true:

$$
\begin{array}{ll}
 & \text{Invariants} \\
 & \text{Guards of the event} \\
\Rightarrow & \\
 & \exists v' \cdot \text{before-after-predicate}
\end{array}
$$

Feasibility Proof Obligation FIS

The action of an event is is possible if guard is true.

$$\forall \overline{vars}, \overline{prms} \cdot grd_e(\overline{vars}, \overline{prms}) \to \exists \overline{vars'} \cdot bap(\overline{vars}, \overline{vars'}, \overline{prms})$$

Deterministic action: $x := t$

. . . nothing to show

Indeterministic action: $x :\in S$

. . . show that $S \neq \varnothing$

Indeterministic action: $x :| \varphi$

. . . show satisfiability of φ

Thus impossible evolutions like $x :| \mathit{false}$ or $x :\in \varnothing$ are avoided

Deadlock Freedom DLKF

Recap:

Deadlock: no event e can be triggered, i.e.
$\forall prms_e \cdot \neg grd_e(t(n), prms_e)$ for all events e.

Deadlock Freedom DLKF

Recap:

Deadlock: no event e can be triggered, i.e.
$\forall prms_e \cdot \neg grd_e(t(n), prms_e)$ for all events e.

Proof Obligation

There is always an event that can trigger:

$$\forall \overline{vars} \cdot inv(\overline{vars}) \Rightarrow \bigvee_{\text{event } e \in M} \exists \overline{prms} \cdot grd_e(\overline{vars}, \overline{prms})$$

Deadlock Freedom DLKF

Recap:

Deadlock: no event e can be triggered, i.e.
$\forall prms_e \cdot \neg grd_e(t(n), prms_e)$ for all events e.

Proof Obligation

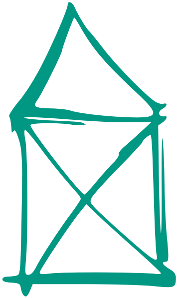
There is always an event that can trigger:

$$\forall \overline{vars} \cdot inv(\overline{vars}) \Rightarrow \bigvee_{\text{event } e \in M} \exists \overline{prms} \cdot grd_e(\overline{vars}, \overline{prms})$$

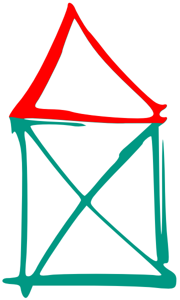
Again, this is sufficient not necessary.
(The invariant may be too weak to imply deadlock freedom)

Event-B –
Refinement

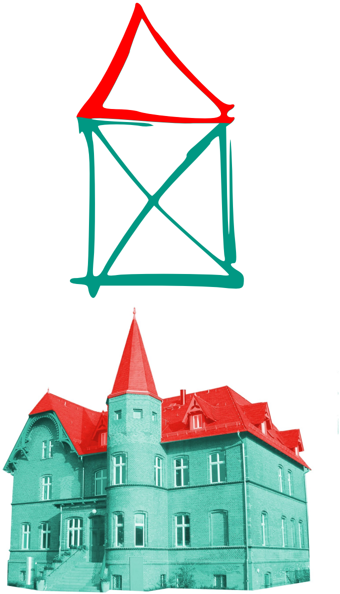
Refinement in Event-B



Refinement in Event-B



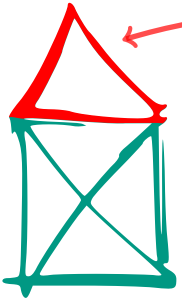
Refinement in Event-B



MACHINE *Abstract*
VARIABLES x
INVARIANTS $x \geq 0$

EVENTS INCREASE $\widehat{=}$
$$x :\mid x' \geq x$$

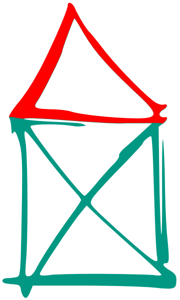
Refinement in Event-B



MACHINE *Abstract*
VARIABLES x
INVARIANTS $x \geq 0$

EVENTS INCREASE $\widehat{=}$
$$x :\mid x' \geq x$$

Refinement in Event-B

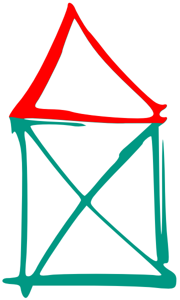


MACHINE *Abstract*
VARIABLES x
INVARIANTS $x \geq 0$

EVENTS INCREASE $\widehat{=}$
$x :| \; x' \geq x$

MACHINE *Refined*
REFINES *Abstract*
VARIABLES x

Refinement in Event-B



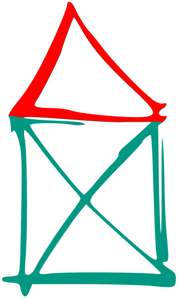
MACHINE *Abstract*
VARIABLES x
INVARIANTS $x \geq 0$

EVENTS $\textsc{Increase} \mathrel{\widehat{=}}$
$x :\mid x' \geq x$

MACHINE *Refined*
REFINES *Abstract*
VARIABLES x

EVENTS $\textsc{NextVal} \mathrel{\widehat{=}}$
REFINES $\textsc{Increase}$
$x := 5 * x^2 + 3 * x$

Refinement in Event-B



MACHINE *Abstract*
VARIABLES x
INVARIANTS $x \geq 0$

EVENTS INCREASE $\widehat{=}$
$x :\mid x' \geq x$

MACHINE *Refined*
REFINES *Abstract*
VARIABLES x

EVENTS NEXTVAL $\widehat{=}$
REFINES INCREASE
$x := 5 * x^2 + 3 * x$

Refining Machines

MACHINE *Abstract*

SEES *Context*

VARIABLES $\overline{vars_A}$

INVARIANTS

$inv_A(\overline{vars_A})$

EVENTS

INITIALISATION $\widehat{=} \ldots$

$\mathrm{EVT}_A \widehat{=} \ldots$

END

MACHINE *Refined*

REFINES *Abstract*

SEES *Context*

VARIABLES $\overline{vars_R}$

INVARIANTS

$inv_R(\overline{vars_A}, \overline{vars_R})$

EVENTS

INITIALISATION $\widehat{=} \ldots$

$\mathrm{EVT}_R \widehat{=}$

REFINES $\mathrm{EVT}_A \ldots$

END

Machines as Relations

Every machine M defines:

- a state space S_M spanned by the types of $vars_M$
- the initialisation $I_M \subseteq S_M$
- the transition relations $E_{M;evt} \in S_M \leftrightarrow S_M$ (for event evt)

Details

$$S_M \quad = \quad \tau(v_1) \times \ldots \times \tau(v_k) \qquad \text{(with } vars_M = v_1, \ldots, v_k\text{)}$$

Machines as Relations

Every machine M defines:

- a state space S_M spanned by the types of $vars_M$
- the initialisation $I_M \subseteq S_M$
- the transition relations $E_{M;evt} \in S_M \leftrightarrow S_M$ (for event evt)

Details

$$\begin{aligned} S_M &= \tau(v_1) \times \ldots \times \tau(v_k) \qquad \text{(with } vars_M = v_1, \ldots, v_k\text{)} \\ I_M(p) &= \{s \in S_M \mid grd_{init}(p) \wedge bap_{init}(s', p)\} \end{aligned}$$

Machines as Relations

Every machine M defines:

- a state space S_M spanned by the types of $vars_M$
- the initialisation $I_M \subseteq S_M$
- the transition relations $E_{M;evt} \in S_M \leftrightarrow S_M$ (for event evt)

Details

$$
\begin{aligned}
S_M &= \tau(v_1) \times \ldots \times \tau(v_k) \qquad \text{(with } vars_M = v_1, \ldots, v_k\text{)} \\
I_M(p) &= \{s \in S_M \mid grd_{init}(p) \wedge bap_{init}(s', p)\} \\
I_M &= \bigcup_p I_M(p)
\end{aligned}
$$

Machines as Relations

Every machine M defines:

- a state space S_M spanned by the types of $vars_M$
- the initialisation $I_M \subseteq S_M$
- the transition relations $E_{M;evt} \in S_M \leftrightarrow S_M$ (for event evt)

Details

$$
\begin{aligned}
S_M &= \tau(v_1) \times \ldots \times \tau(v_k) \qquad \text{(with } vars_M = v_1, \ldots, v_k\text{)} \\
I_M(p) &= \{s \in S_M \mid grd_{init}(p) \wedge bap_{init}(s', p)\} \\
I_M &= \bigcup_p I_M(p) \\
E_{M;evt}(p) &= \{(s \mapsto s') \mid grd_{evt}(s, p) \wedge bap_{evt}(s, s', p)\}
\end{aligned}
$$

Machines as Relations

Every machine M defines:

- a state space S_M spanned by the types of $vars_M$
- the initialisation $I_M \subseteq S_M$
- the transition relations $E_{M;evt} \in S_M \leftrightarrow S_M$ (for event evt)

Details

$$
\begin{aligned}
S_M &= \tau(v_1) \times \ldots \times \tau(v_k) \qquad \text{(with } vars_M = v_1, \ldots, v_k\text{)} \\
I_M(p) &= \{s \in S_M \mid grd_{init}(p) \land bap_{init}(s', p)\} \\
I_M &= \bigcup_p I_M(p) \\
E_{M;evt}(p) &= \{(s \mapsto s') \mid grd_{evt}(s, p) \land bap_{evt}(s, s', p)\} \\
E_{M;evt} &= \bigcup_p E_{M;evt}(p)
\end{aligned}
$$

Simple Refinement – Definition

Every trace of the refined machine R is
a trace of the abstract machine A.

Definition: Simple Refinement

Let R, A be two machines with the same state space S.
R is called a refinement of A if

1. $I_R \subseteq I_A$ and

Simple Refinement – Definition

Every trace of the refined machine R is
a trace of the abstract machine A.

Definition: Simple Refinement

Let R, A be two machines with the same state space S.
R is called a refinement of A if

1. $I_R \subseteq I_A$ and
2. $E_{R;evt_R} \subseteq E_{A;evt_A}$ for each event

Simple Refinement – Definition

Every trace of the refined machine R is
a trace of the abstract machine A.

Definition: Simple Refinement

Let R, A be two machines with the same state space S.
R is called a refinement of A if

1. $I_R \subseteq I_A$ and
2. $E_{R;evt_R} \subseteq E_{A;evt_A}$ for each event

(evt_R is the event in R that refines event evt_A from A)

Loss of behaviour

Why is this problematic?

MACHINE A $\quad \ldots$
EVENT $emergencyStop \mathrel{\widehat{=}}$
WHERE $true$ THEN $heavyMachine := stop$
END

refined by

MACHINE R $\quad \ldots$
EVENT $emergencyStop \mathrel{\widehat{=}}$ REFINES $emergencyStop$
WHERE $false$ THEN $heavyMachine := stop$
END

Loss of behaviour

Why is this problematic?

MACHINE A $\quad \ldots$
EVENT$\mathit{emergencyStop} \mathrel{\widehat{=}}$
WHERE true THEN $\mathit{heavyMachine} := \mathit{stop}$
END

refined by

MACHINE R $\quad \ldots$
EVENT$\mathit{emergencyStop} \mathrel{\widehat{=}}$ REFINES $\mathit{emergencyStop}$
WHERE false THEN $\mathit{heavyMachine} := \mathit{stop}$
END

Loss of behaviour

Why is this problematic?

MACHINE A $\quad \ldots$
EVENT $emergencyStop \mathrel{\widehat{=}}$
WHERE $true$ THEN $heavyMachine := stop$
END

refined by

MACHINE R $\quad \ldots$
EVENT $emergencyStop \mathrel{\widehat{=}}$ REFINES $emergencyStop$
WHERE $false$ THEN $heavyMachine := stop$
END

$E_{R;evt} = \varnothing \implies R$ refines A

Loss of behaviour

Every trace for A has a refining trace for R.

Loss of behaviour

Every trace for A has a refining trace for R.

More precisely

For every trace in A with triggered events $evt_{A,1}, evt_{A,2}, \ldots$, there is a trace in R with triggered events $evt_{R,1}, evt_{R,2}, \ldots$ and $evt_{R;i}$ refines $evt_{A;i}$.

Loss of behaviour

Every trace for A has a refining trace for R.

More precisely

For every trace in A with triggered events $evt_{A,1}, evt_{A,2}, \ldots$, there is a trace in R with triggered events $evt_{R,1}, evt_{R,2}, \ldots$ and $evt_{R;i}$ refines $evt_{A;i}$.

Definition: Lockfree Refinement

Let R, A be two machines with the same state space S.
R is called a *lockfree* refinement of A if

1. $I_R \subseteq I_A$

Loss of behaviour

Every trace for A has a refining trace for R.

More precisely

For every trace in A with triggered events $evt_{A,1}, evt_{A,2}, \ldots$, there is a trace in R with triggered events $evt_{R,1}, evt_{R,2}, \ldots$ and $evt_{R;i}$ refines $evt_{A;i}$.

Definition: Lockfree Refinement

Let R, A be two machines with the same state space S.
R is called a *lockfree* refinement of A if

1. $I_R \subseteq I_A$
2. $I_R \neq \varnothing$

Loss of behaviour

Every trace for A has a refining trace for R.

More precisely

For every trace in A with triggered events $evt_{A,1}, evt_{A,2}, \ldots$, there is a trace in R with triggered events $evt_{R,1}, evt_{R,2}, \ldots$ and $evt_{R;i}$ refines $evt_{A;i}$.

Definition: Lockfree Refinement

Let R, A be two machines with the same state space S.
R is called a *lockfree* refinement of A if

1. $I_R \subseteq I_A$
2. $I_R \neq \varnothing$
3. $E_{R;evt_R} \subseteq E_{A;evt_A}$ for each event

Loss of behaviour

Every trace for A has a refining trace for R.

More precisely

For every trace in A with triggered events $evt_{A,1}, evt_{A,2}, \ldots$, there is a trace in R with triggered events $evt_{R,1}, evt_{R,2}, \ldots$ and $evt_{R;i}$ refines $evt_{A;i}$.

Definition: Lockfree Refinement

Let R, A be two machines with the same state space S.
R is called a *lockfree* refinement of A if

1. $I_R \subseteq I_A$
2. $I_R \neq \varnothing$
3. $E_{R;evt_R} \subseteq E_{A;evt_A}$ for each event
4. $\mathrm{dom}(E_{A;evt_A}) \subseteq \mathrm{dom}(E_{R;evt_R})$ for each event

Coupling

More general notion of refinement

What if abstract machine A and refinement R have different state spaces S_A and S_R?

Coupling

More general notion of refinement

What if abstract machine A and refinement R have different state spaces S_A and S_R?

➜ **Couple** abstract and refined state space.

$C \in S_R \leftrightarrow S_A$ **Coupling invariant / Gluing invariant**

Coupling

More general notion of refinement

What if abstract machine A and refinement R have different state spaces S_A and S_R?

→ **Couple** abstract and refined state space.

$C \in S_R \leftrightarrow S_A$ **Coupling invariant / Gluing invariant**

Example

MACHINE *AbstractFileSys*
VARIABLES *openFiles*
INVARIANTS
$openFiles \subseteq FILES$

MACHINE *RefinedFileSys*
VARIABLES *openModes*
INVARIANTS
$openModes \subseteq FILES \times MODES$

Coupling

More general notion of refinement

What if abstract machine A and refinement R have different state spaces S_A and S_R?

➜ **Couple** abstract and refined state space.

$C \in S_R \leftrightarrow S_A$ **Coupling invariant / Gluing invariant**

Example

MACHINE *AbstractFileSys*
VARIABLES *openFiles*
INVARIANTS
$openFiles \subseteq FILES$

MACHINE *RefinedFileSys*
VARIABLES *openModes*
INVARIANTS
$openModes \subseteq FILES \times MODES$

$$C = \{r \mapsto a \mid a = \mathrm{dom}(r)\} = \{f, m \cdot (f \mapsto m) \mapsto m\}$$

Refinement – Coupling

- Sensible to assume C a total relation:

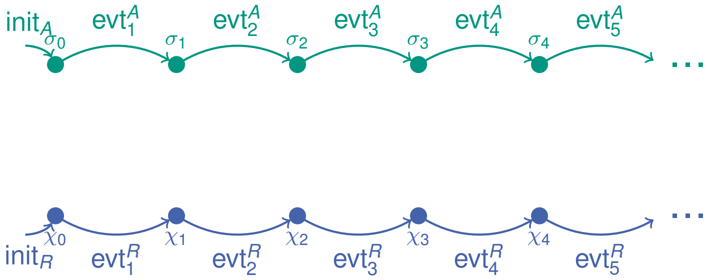
$$C \in S_R \twoheadleftarrow\!\!\!\rightarrow S_A$$

- Often, coupling is a total function:

$$C \in S_R \rightarrow S_A$$

Define *one* abstraction for any detailed state.
BUT sometimes, several possible abstractions per concrete state sensible.

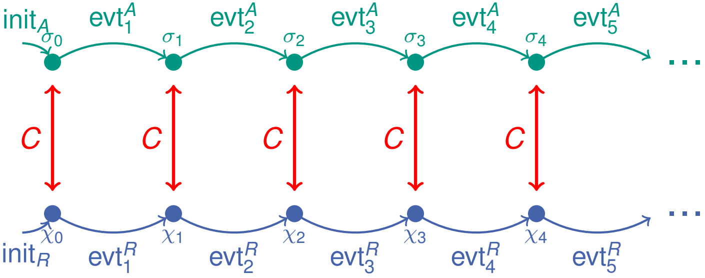
Refinement – Coupled Traces



Refinement – Coupled Traces

$$
\begin{array}{ccccccccccc}
\xrightarrow{\text{init}_A} & \sigma_0 & \xrightarrow{\text{evt}_1^A} & \sigma_1 & \xrightarrow{\text{evt}_2^A} & \sigma_2 & \xrightarrow{\text{evt}_3^A} & \sigma_3 & \xrightarrow{\text{evt}_4^A} & \sigma_4 & \xrightarrow{\text{evt}_5^A} \cdots \\
 & C \updownarrow & & C \updownarrow & & C \updownarrow & & C \updownarrow & & C \updownarrow & \\
\xrightarrow{\text{init}_R} & \chi_0 & \xrightarrow{\text{evt}_1^R} & \chi_1 & \xrightarrow{\text{evt}_2^R} & \chi_2 & \xrightarrow{\text{evt}_3^R} & \chi_3 & \xrightarrow{\text{evt}_4^R} & \chi_4 & \xrightarrow{\text{evt}_5^R} \cdots
\end{array}
$$

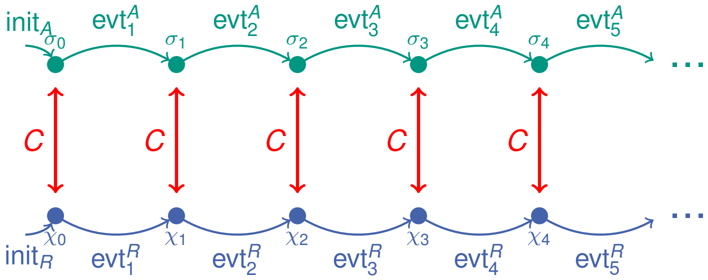
Refinement – Coupled Traces



Refinement: *R* refines *A*

For every concrete trace $(\chi_0, \chi_1, \ldots)$ of R with events $(evt_1^R, evt_2^R, ...)$ there exists an abstract trace $(\sigma_0, \sigma_1, \ldots)$ with events $(evt_1^A, evt_2^A, \ldots)$ such that

Refinement – Coupled Traces

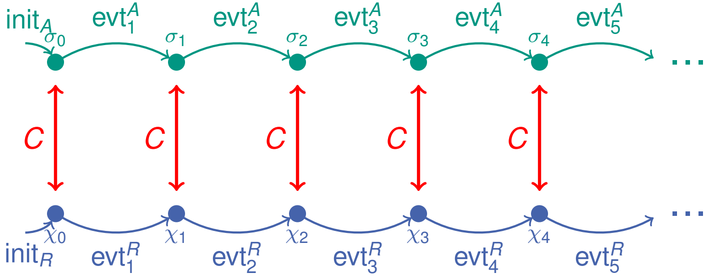


Refinement: *R* refines *A*

For every concrete trace $(\chi_0, \chi_1, \ldots)$ of R with events $(evt_1^R, evt_2^R, ...)$ there exists an abstract trace $(\sigma_0, \sigma_1, \ldots)$ with events $(evt_1^A, evt_2^A, \ldots)$ such that

1. $\chi_i \mapsto \sigma_i \in C$ for all $i \in \mathbb{N}$

Refinement – Coupled Traces



Refinement: *R* refines *A*

For every concrete trace $(\chi_0, \chi_1, \ldots)$ of R with events $(evt_1^R, evt_2^R, ...)$ there exists an abstract trace $(\sigma_0, \sigma_1, \ldots)$ with events $(evt_1^A, evt_2^A, \ldots)$ such that

1. $\chi_i \mapsto \sigma_i \in C$ for all $i \in \mathbb{N}$
2. evt_i^R refines event evt_i^A.

Refinement – Definition

Definition: Refinement

Let R, A be two machines with state spaces S_R, S_A.
Let $C \in S_R \leftrightarrow R_A$ be the coupling invariant.
R is called a refinement of A modulo C if

Refinement – Definition

Definition: Refinement

Let R, A be two machines with state spaces S_R, S_A.
Let $C \in S_R \leftrightarrow R_A$ be the coupling invariant.
R is called a refinement of A modulo C if

1. $I_R \subseteq C^{-1}[I_A]$ and

($\forall x, y \cdot x \mapsto y \in R^{-1} \Leftrightarrow y \mapsto x \in R$, inverse relation)

Refinement – Definition

Definition: Refinement

Let R, A be two machines with state spaces S_R, S_A.
Let $C \in S_R \leftrightarrow R_A$ be the coupling invariant.
R is called a refinement of A modulo C if

1. $I_R \subseteq C^{-1}[I_A]$ and
2. $E_{R;evt_R} \subseteq C \mathbin{;} E_{A;evt_A} \mathbin{;} C^{-1}$ for each event.

($\forall x, y \cdot x \mapsto y \in R^{-1} \Leftrightarrow y \mapsto x \in R$, inverse relation)

Refinement – Path subsumption

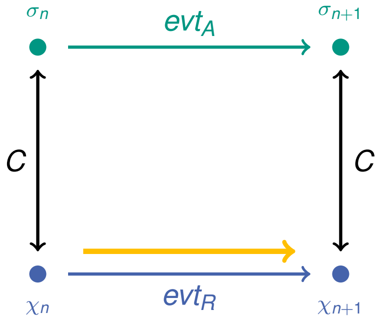
$$
\begin{tikzcd}
\sigma_n \arrow[r, "evt_A"] \arrow[d, leftrightarrow, "C"'] & \sigma_{n+1} \arrow[d, leftrightarrow, "C"] \\
\chi_n \arrow[r, "evt_R"'] & \chi_{n+1}
\end{tikzcd}
$$

Refinement – Path subsumption

$$
\begin{tikzcd}
\sigma_n \arrow[r, "evt_A"] \arrow[d, leftrightarrow, "C"'] & \sigma_{n+1} \arrow[d, leftrightarrow, "C"] \\
\chi_n \arrow[r, "evt_R"'] & \chi_{n+1}
\end{tikzcd}
$$

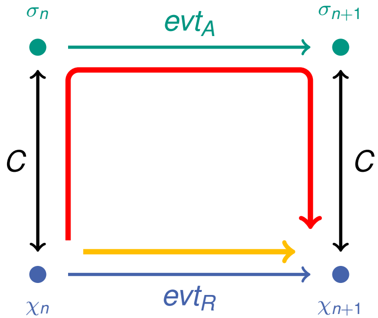
$$E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$$

Refinement – Path subsumption



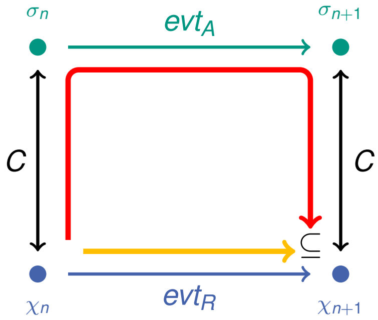
$$E_{R;evt_R} \subseteq C \, ; E_{A;evt_A} \, ; C^{-1}$$

Refinement – Path subsumption



$$E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$$

Refinement – Path subsumption



$$E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$$

Specifying Coupling

The coupling invariant is specified as
part of the invariant of the refining machine.

Specifying Coupling

The coupling invariant is specified as
part of the invariant of the refining machine.

The invariant of a refinement is allowed to refer to variables of its abstraction.

Specifying Coupling

The coupling invariant is specified as
part of the invariant of the refining machine.

The invariant of a refinement is allowed to refer to variables of its abstraction.

Example (from slide 72)

MACHINE *AbstractFileSys* VARIABLES *openFiles* INVARIANTS $openFiles \subseteq FILES$	MACHINE *RefinedFileSys* VARIABLES *openModes* INVARIANTS $openModes \subseteq FILES \times MODES$

Specifying Coupling

The coupling invariant is specified as
part of the invariant of the refining machine.

The invariant of a refinement is allowed to refer to variables of its abstraction.

Example (from slide 72)

MACHINE *AbstractFileSys* VARIABLES *openFiles* INVARIANTS $openFiles \subseteq FILES$	MACHINE *RefinedFileSys* VARIABLES *openModes* INVARIANTS $openModes \subseteq FILES \times MODES$ $openFiles = \mathrm{dom}(openModes)$

Proof Obligation GRD

Proof that event guard in refinement is **stronger** than in abstract machine.

$\Longrightarrow$ Abstraction is enabled when refinement is.

$$
\begin{array}{ll}
 & \text{Abstract invariants} \\
 & \text{Concrete invariants} \\
 & \text{Concrete event guard} \\
\Longrightarrow & \\
 & \text{Abstract event guard}
\end{array}
$$

Proof Obligation GRD

Proof that event guard in refinement is **stronger** than in abstract machine.
$\Longrightarrow$ Abstraction is enabled when refinement is.

	Abstract invariants Concrete invariants Concrete event guard
$\Longrightarrow$	
	Abstract event guard

$$\forall \overline{vars_A}, \overline{vars_R} \cdot$$

$$inv_A(\overline{vars_A}) \wedge inv_R(\overline{vars_A}, \overline{vars_R}) \wedge grd_R(\overline{vars_R})$$

$$\Rightarrow grd_A(\overline{vars_A})$$

(Version w/o parameters, see literature for full version)

Proof Obligation SIM

Show that refined action *simulates* abstract actions

$$
\begin{array}{ll}
 & \text{Abstract invariants} \\
 & \text{Concrete invariants} \\
 & \text{Concrete event guard} \\
 & \text{Concrete before-after-predicate} \\
\Longrightarrow & \\
 & \text{Abstract before-after-predicate}
\end{array}
$$

Proof Obligation SIM

Show that refined action *simulates* abstract actions

$$
\begin{array}{ll}
 & \text{Abstract invariants} \\
 & \text{Concrete invariants} \\
 & \text{Concrete event guard} \\
 & \text{Concrete before-after-predicate} \\
\Longrightarrow & \\
 & \text{Abstract before-after-predicate}
\end{array}
$$

Rem $E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$

Proof Obligation SIM

Show that refined action *simulates* abstract actions

$$
\begin{array}{ll}
 & \text{Abstract invariants} \\
 & \text{Concrete invariants} \\
 & \text{Concrete event guard} \\
 & \text{Concrete before-after-predicate} \\
\Longrightarrow & \\
 & \text{Abstract before-after-predicate}
\end{array}
$$

Rem $E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$

Obs The coupling invariant is only used for the before-state not for the after-state.

Proof Obligation SIM

Show that refined action *simulates* abstract actions

$$
\begin{array}{l}
\quad\text{Abstract invariants} \\
\quad\text{Concrete invariants} \\
\quad\text{Concrete event guard} \\
\quad\text{Concrete before-after-predicate} \\
\Longrightarrow \\
\quad\text{Abstract before-after-predicate}
\end{array}
$$

Rem $E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$

Obs The coupling invariant is only used for the before-state not for the after-state.

? Why?

Proof Obligation SIM

Show that refined action *simulates* abstract actions

$$\begin{array}{ll} & \text{Abstract invariants} \\ & \text{Concrete invariants} \\ & \text{Concrete event guard} \\ & \text{Concrete before-after-predicate} \\ \Longrightarrow & \\ & \text{Abstract before-after-predicate} \end{array}$$

Rem $E_{R;evt_R} \subseteq C \,;\, E_{A;evt_A} \,;\, C^{-1}$

Obs The coupling invariant is only used for the before-state not for the after-state.

? Why?

! Already proven condition INV implies invariant for after-state.

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Event-B has more ...

Things not covered in these slides:

- Witnesses for parameters dropped in refinements
- Termination issues (variants)
- Extended/Not extended events
- Event merging
- Sequential refinement
- ...

Byzantine Agreement –
A case study verified with Event-B

Based on:

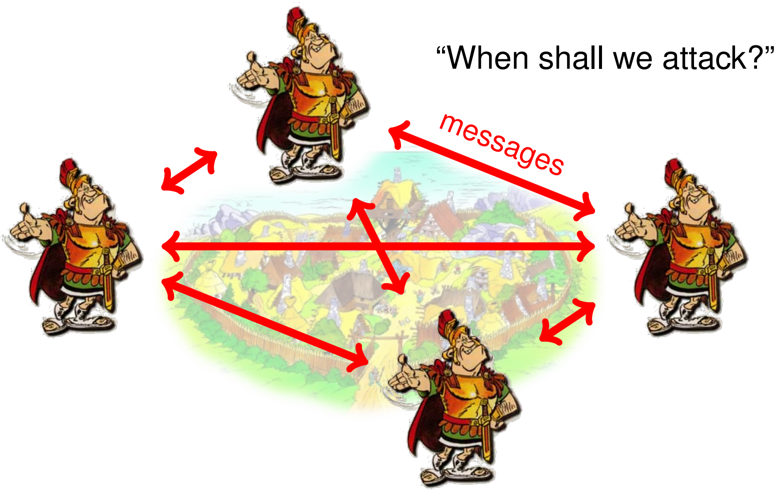
Roman Krenický and Mattias Ulbrich. *Deductive Verification of a Byzantine Agreement Protocol*. Technical report (2010-7). Karlsruhe Institute of Technology, Department of Informatics, 2010

Byzantine Generals

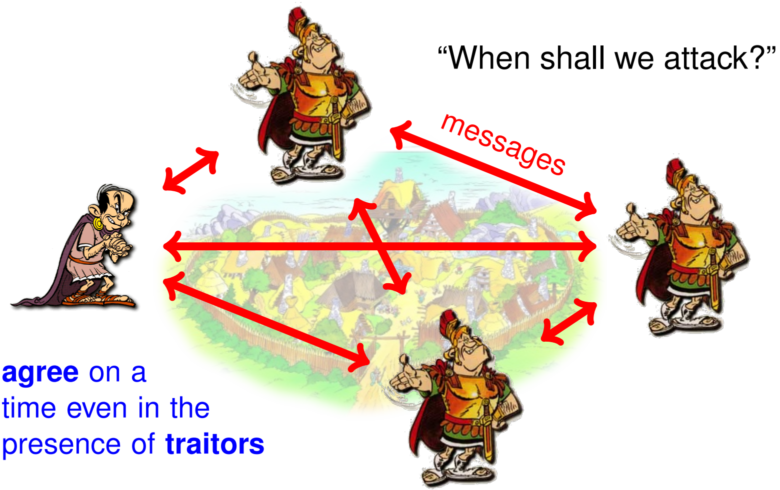
Byzantine Generals

“When shall we attack?”

Byzantine Generals

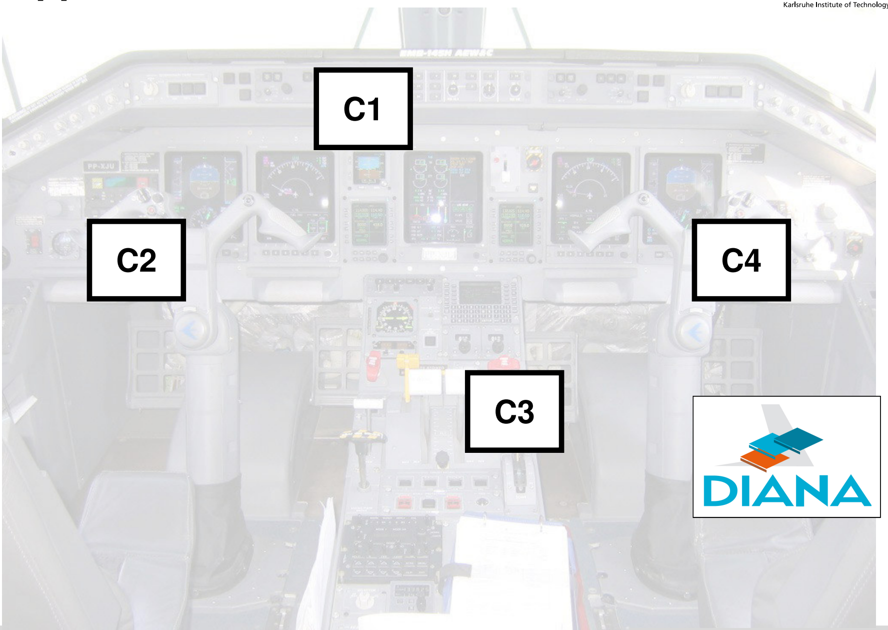


Byzantine Generals

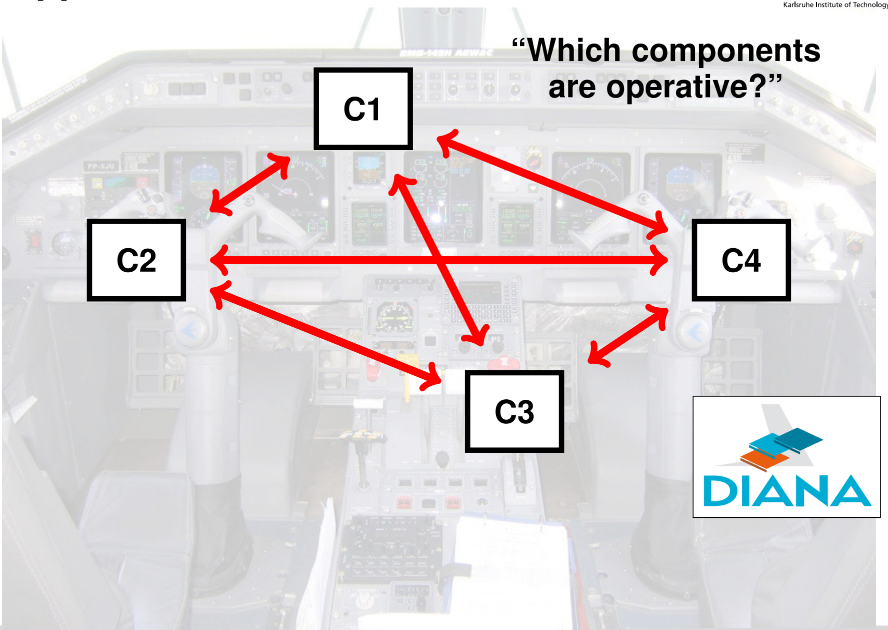


agree on a time even in the presence of **traitors**

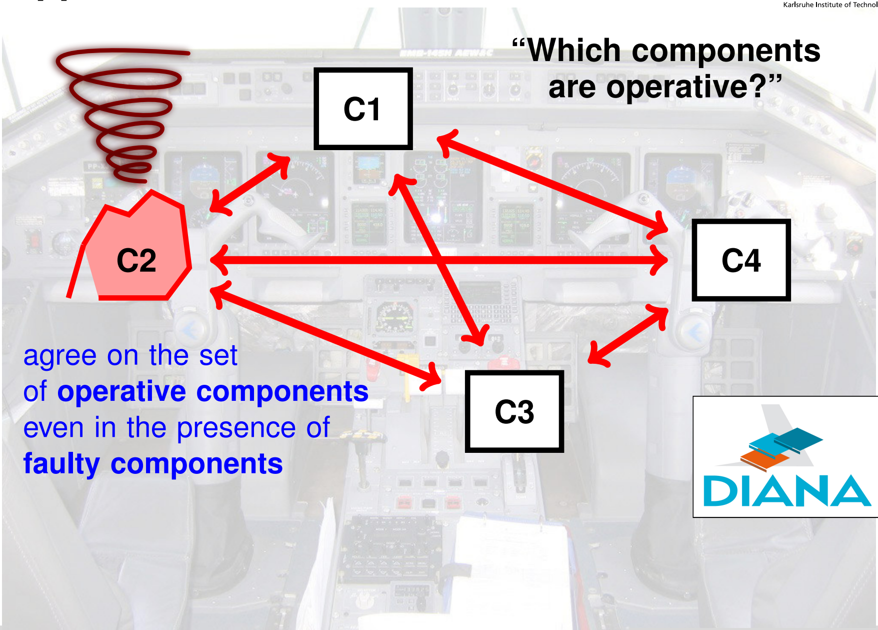
Application in Avionics



Application in Avionics



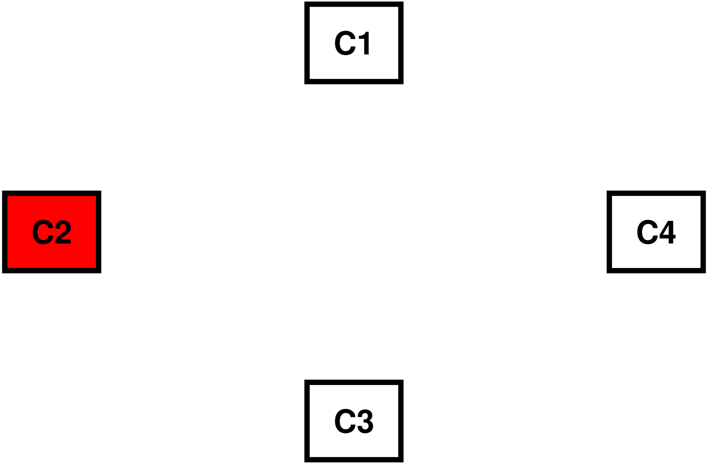
Application in Avionics



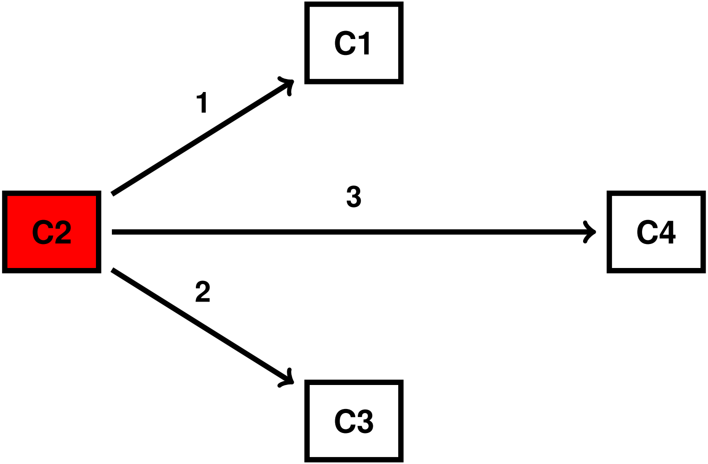
"Which components are operative?"

agree on the set
of **operative components**
even in the presence of
faulty components

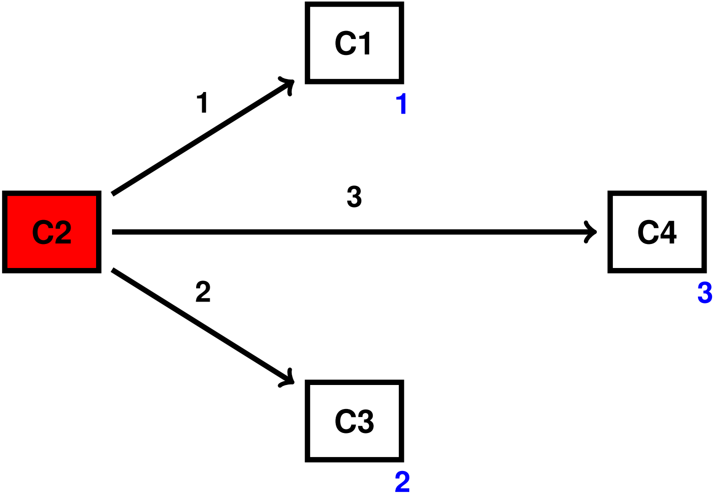
Explanation by Example



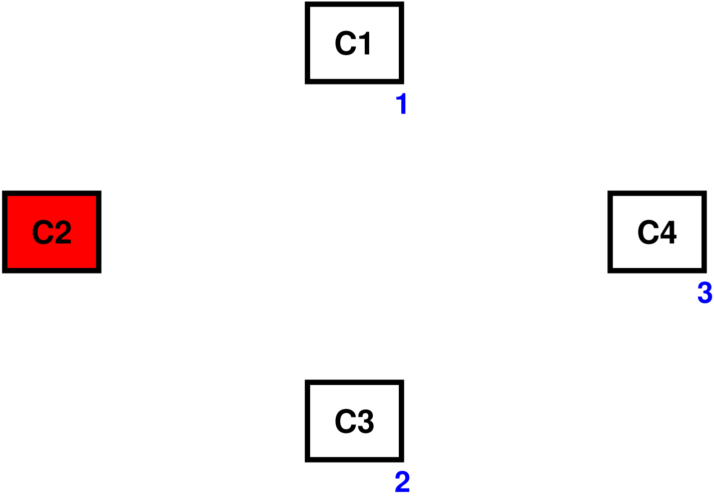
Explanation by Example



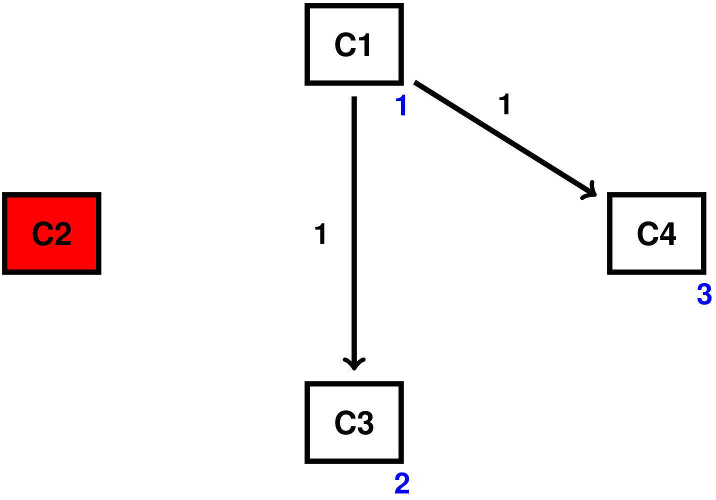
Explanation by Example



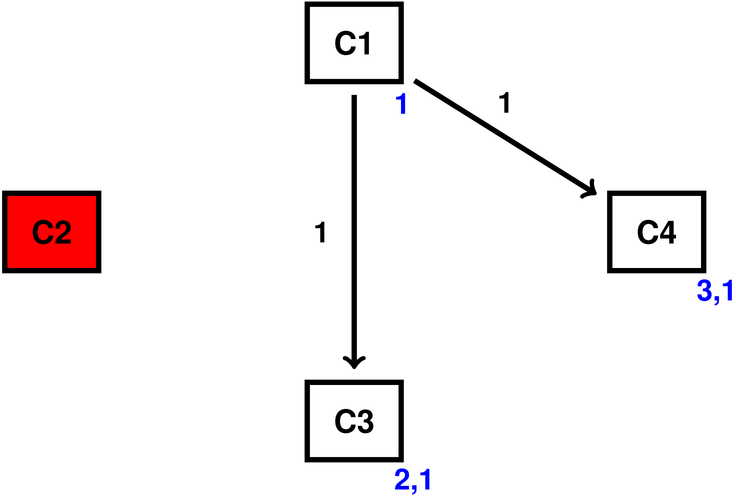
Explanation by Example



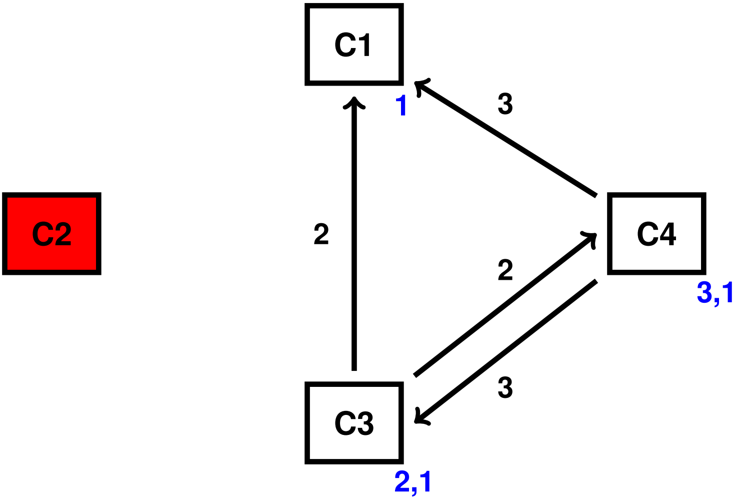
Explanation by Example



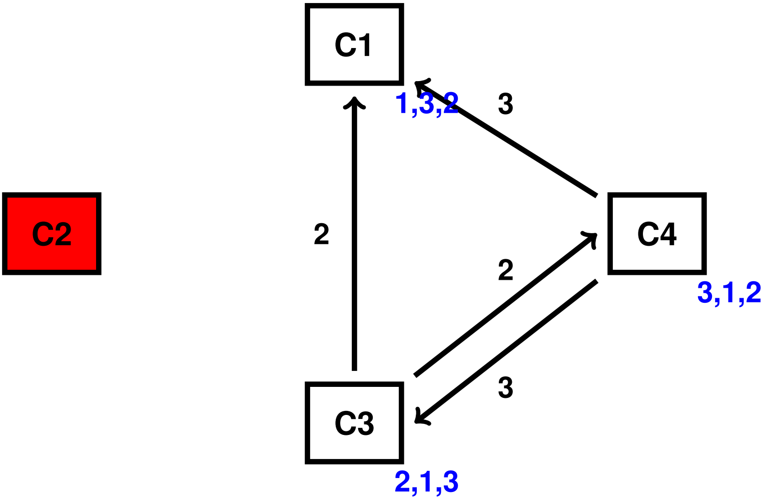
Explanation by Example



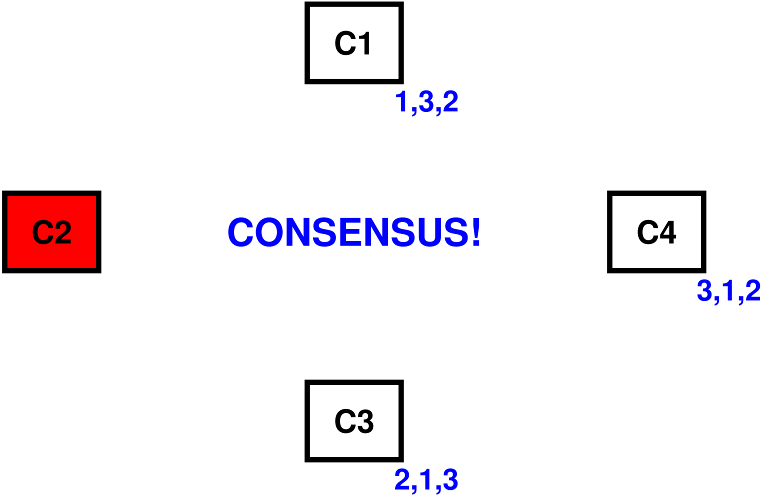
Explanation by Example



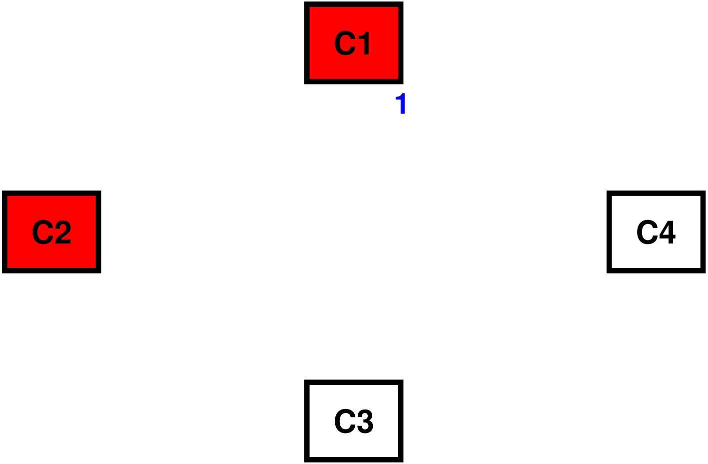
Explanation by Example



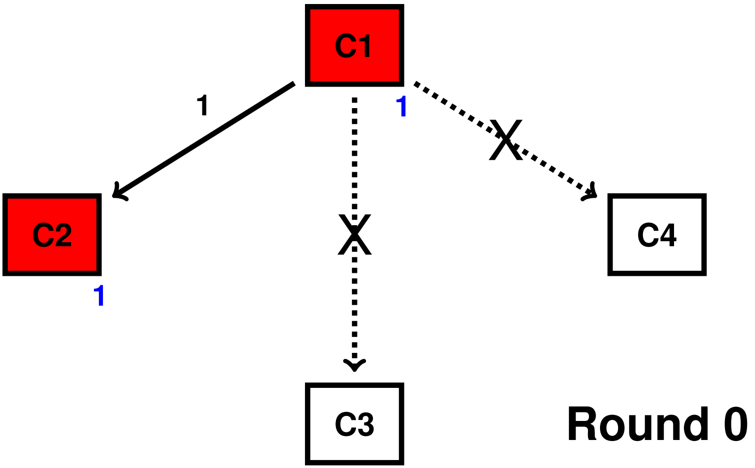
Explanation by Example



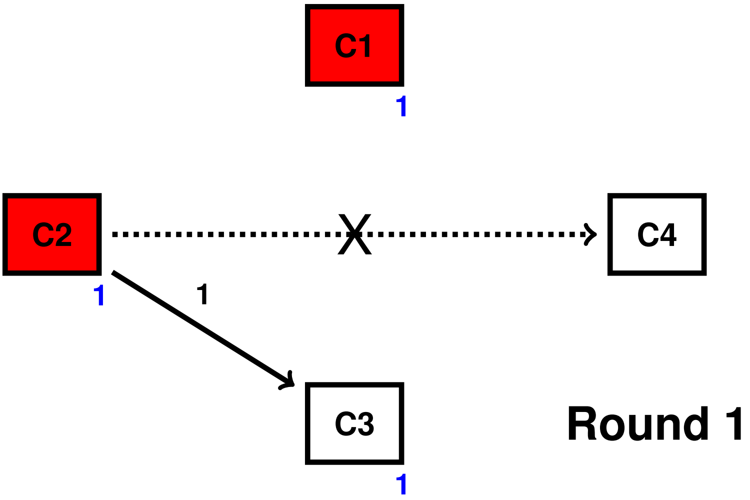
Example Run 2



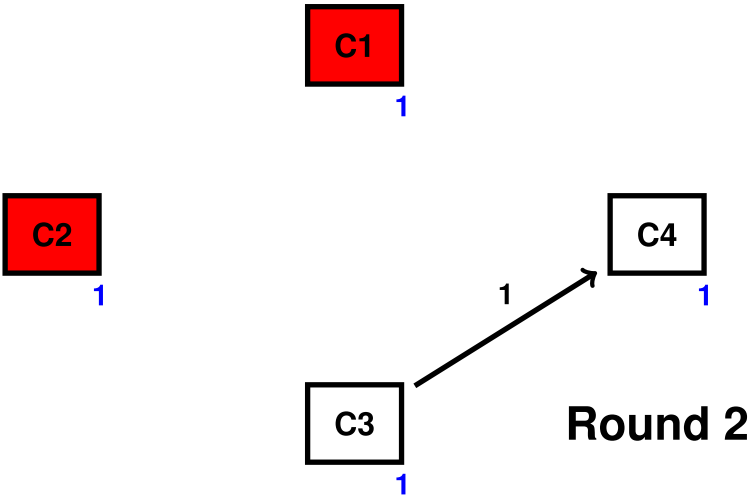
Example Run 2



Example Run 2



Example Run 2



Byzantine Agreement Algorithm

Verification Goals:

Validity If the transmitter *tt* is non-faulty, then all non-faulty receivers agree on the value sent by *tt*.

Agreement Any two non-faulty receivers agree on the value assigned to *tt*.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round n: Component receive messages, verify signatures, sign messages and pass them on.

Byzantine Agreement Algorithm

Round 0: Transmitter sends signed message to all receivers.

Round *n*: Component receive messages, verify signatures, sign messages and pass them on.

GOAL: Prove that this algorithm has the “validity” and “agreement” properties.

Verification

Quote

We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm.

Taken from: *The Byzantine Generals Problem*

Leslie Lamport, Robert Shostak, and Marshall Pease
ACM Transactions on Programming Languages and Systems
Volume 4, pp. 383–401,1982.

Context for Byzantine Agreement

CONTEXT *Context*
SETS

CONSTANTS

AXIOMS

END

Context for Byzantine Agreement

```
CONTEXT Context
SETS
    MODULE
    VALUE
CONSTANTS

AXIOMS




END
```

Context for Byzantine Agreement

CONTEXT *Context*
SETS
  MODULE
  VALUE
CONSTANTS
  faulty, *transmitter*, V_0
AXIOMS

END

Context for Byzantine Agreement

CONTEXT *Context*
SETS
$\quad$ $\textsc{Module}$
$\quad$ $\textsc{Value}$
CONSTANTS
$\quad$ $faulty, transmitter, V_0$
AXIOMS
$\quad$ $faulty \subseteq \textsc{Module}$
$\quad$ $transmitter \in \textsc{Module}$
$\quad$ $V_0 \in \textsc{Value}$
$\quad$ $\mathrm{finite}(faulty)$
END

First machine

Karlsruhe Institute of Technology

MACHINE Messages
SEES Context

VARIABLES

INVARIANTS

...

First machine

MACHINE Messages
SEES Context

VARIABLES *messages*, *round*, *collected*

INVARIANTS

...

First machine

MACHINE Messages
SEES Context

VARIABLES *messages*, *round*, *collected*

INVARIANTS
  ty_mess : $messages \subseteq \textsc{Module} \times \textsc{Module} \times \textsc{Value}$

...

messages messages being sent in the *current* round

First machine

MACHINE Messages
SEES Context

VARIABLES *messages*, *round*, *collected*

INVARIANTS

ty_mess : $messages \subseteq \textsc{Module} \times \textsc{Module} \times \textsc{Value}$
ty_round : $round \in \mathbb{N}$

...

messages messages being sent in the *current* round

round the number of the current round

First machine

MACHINE Messages
SEES Context

VARIABLES *messages*, *round*, *collected*

INVARIANTS

ty_mess : $messages \subseteq \textsc{Module} \times \textsc{Module} \times \textsc{Value}$

ty_round : $round \in \mathbb{N}$

ty_collected : $collected \in \textsc{Module} \to \mathbb{P}(\textsc{Value})$

...

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

$act1 : round := round + 1$

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

$act1$: $round := round + 1$

$act2$: $messages :\in \mathbb{P}(\textsc{Module} \times \textsc{Module} \times \textsc{Value})$

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

$act1 : round := round + 1$

$act2 : messages :\in \mathbb{P}(\textsc{Module} \setminus \{transmitter\} \times \textsc{Module} \times \textsc{Value})$

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

$act1 : round := round + 1$

$act2 : messages :\in \mathbb{P}(\textsc{Module} \setminus \{transmitter\} \times \textsc{Module} \times \textsc{Value})$

$act3 : collected := \lambda m \cdot collected(m) \cup$

END

First machine (2)

messages	messages being sent in the *current* round
round	the number of the current round
collected	values observed in previous rounds

MACHINE *Messages* SEES *Context*

VARIABLES *messages*, *round*, *collected*

INVARIANTS...

EVENTS

Initialisation $\widehat{=}$ ...

EVENT *ROUND* $\widehat{=}$

$act1$: $round := round + 1$

$act2$: $messages :\in \mathbb{P}(\textsc{Module} \setminus \{transmitter\} \times \textsc{Module} \times \textsc{Value})$

$act3$: $collected := \lambda m \cdot collected(m) \cup \{v \mid (s, m, v) \in messages\}$

END

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

round k: $\qquad s \xrightarrow{v} r$

First refinement: signed messages

All messages are signed in a trustworthy manner:

No forgery possible $\Longrightarrow$ Consider only **relayed** messages.

round k: $\quad s \xrightarrow{v} r$

round $k+1$: $\quad r \xrightarrow{v} n$

Signed messages (2)

round k: $s \xrightarrow{v} r$

round $k+1$: $r \xrightarrow{v} n$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

EVENTS

END

Signed messages (2)

round k: $s \xrightarrow{v} r$

round $k+1$: $r \xrightarrow{v} n$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

EVENTS

EVENT *ROUND* REFINES *ROUND* $\widehat{=}$

act1, act3 as above

END

Signed messages (2)

$$\text{round } k: \quad s \xrightarrow{v} r$$
$$\text{round } k+1: \quad r \xrightarrow{v} n$$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

EVENTS

EVENT *ROUND* REFINES *ROUND* $\widehat{=}$

act1, act3 as above

$was : messages :\in \mathbb{P}(\textsc{Module} \setminus \{transmitter\} \times \textsc{Module} \times \textsc{Value})$

END

Signed messages (2)

$$\text{round } k: \quad s \xrightarrow{v} r$$
$$\text{round } k+1: \quad r \xrightarrow{v} n$$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

EVENTS

EVENT *ROUND* REFINES *ROUND* $\widehat{=}$

act1, act3 as above

act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

was : $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

END

Signed messages (2)

round k: $s \xrightarrow{v} r$

round $k+1$: $r \xrightarrow{v} n$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

val1: $\forall s, r, v \cdot (s, r, v) \in messages \Rightarrow v \in collected(transmitter)$

EVENTS

EVENT *ROUND* REFINES *ROUND* $\widehat{=}$

act1, act3 as above

act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

$was : messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

END

Signed messages (2)

round k: $s \xrightarrow{v} r$

round $k+1$: $r \xrightarrow{v} n$

MACHINE *SignedMessages* REFINES *Messages*

VARIABLES messages, round, collected

INVARIANTS

val1: $\forall s, r, v \cdot (s, r, v) \in messages \Rightarrow v \in collected(transmitter)$

val2: $\forall n \cdot collected(n) \subseteq collected(transmitter)$

EVENTS

EVENT *ROUND* REFINES *ROUND* $\widehat{=}$

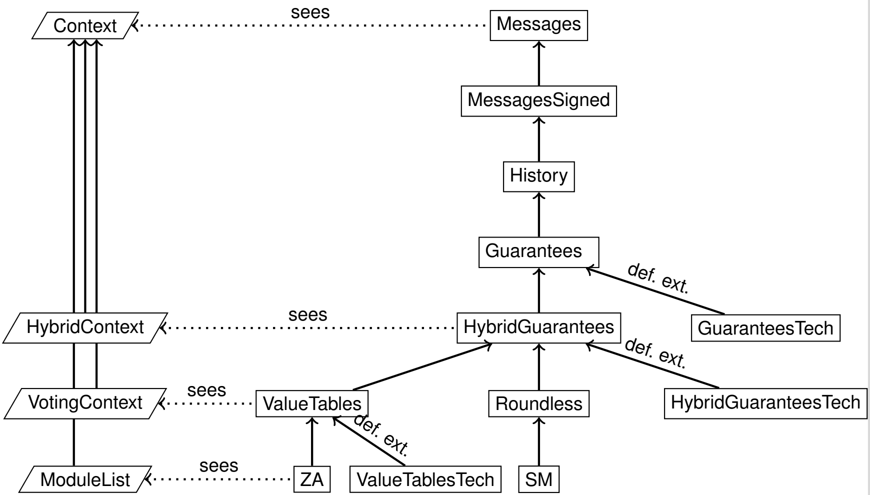
act1, act3 as above

act2: $messages :\in \mathbb{P}\left(\{(r, n, v) \mid (s, r, v) \in messages\}\right)$

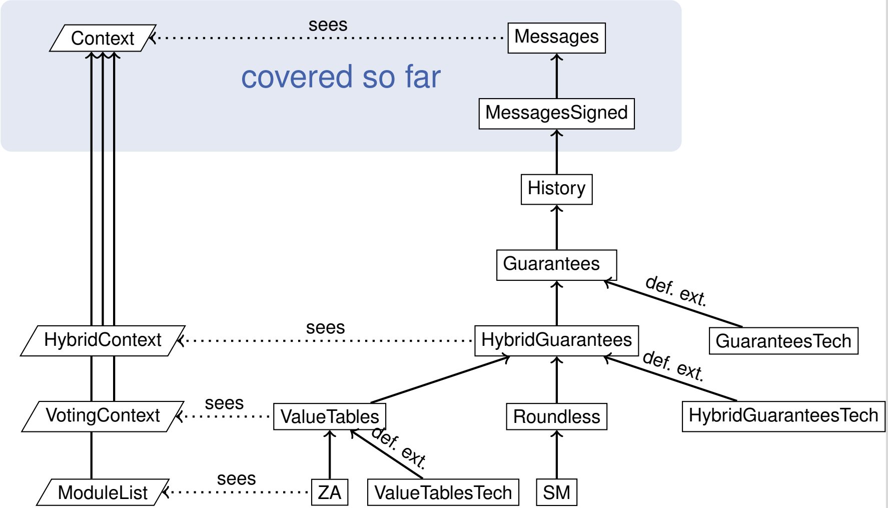
was : $messages :\in \mathbb{P}(\text{MODULE} \setminus \{transmitter\} \times \text{MODULE} \times \text{VALUE})$

END

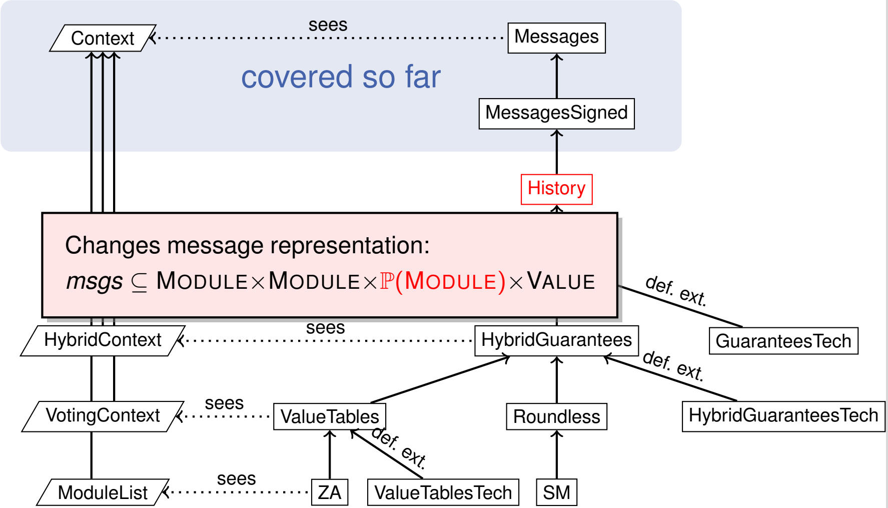
Refinement Tower



Refinement Tower



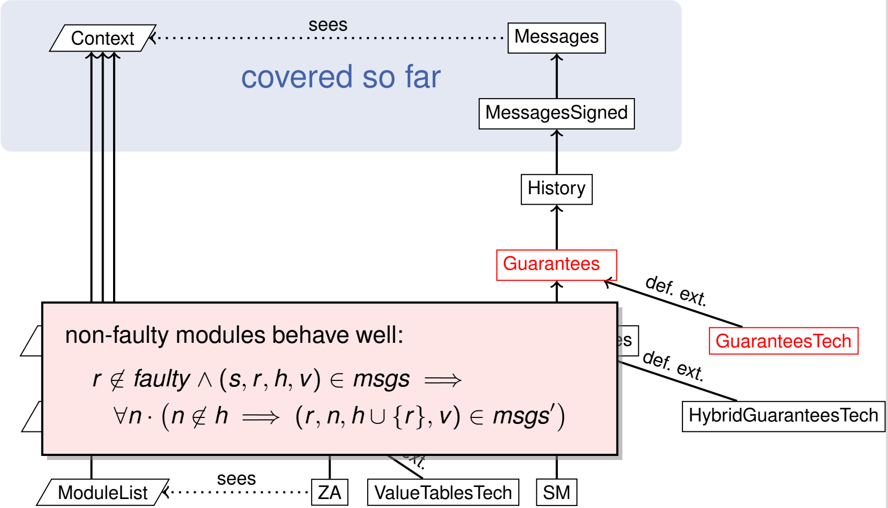
Refinement Tower



Changes message representation:

$msgs \subseteq \textsc{Module} \times \textsc{Module} \times \mathbb{P}(\textsc{Module}) \times \textsc{Value}$

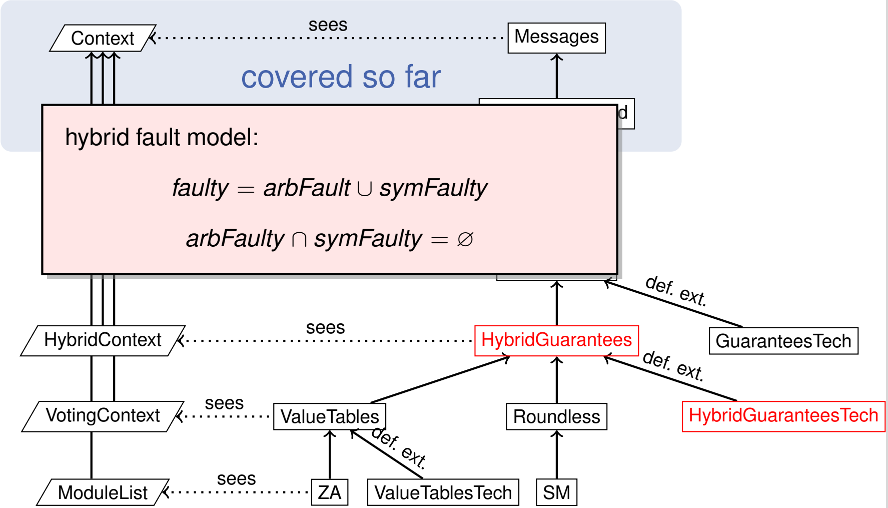
Refinement Tower



non-faulty modules behave well:

$$r \notin faulty \land (s, r, h, v) \in msgs \implies$$
$$\forall n \cdot (n \notin h \implies (r, n, h \cup \{r\}, v) \in msgs')$$

Refinement Tower



hybrid fault model:

$$\mathit{faulty} = \mathit{arbFault} \cup \mathit{symFaulty}$$

$$\mathit{arbFaulty} \cap \mathit{symFaulty} = \varnothing$$

Refinement Tower

new event structure:

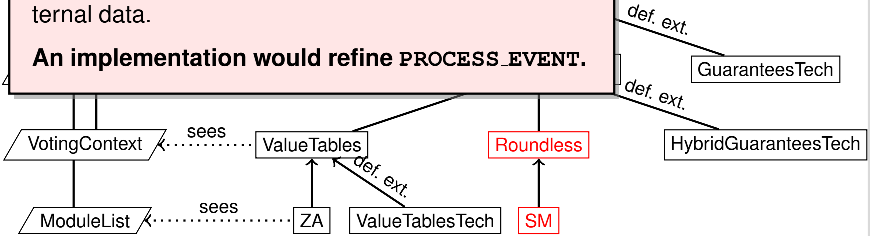
`PROCESS_EVENT refines SKIP`

modifies internal data structures (invisible to abstract machine) and

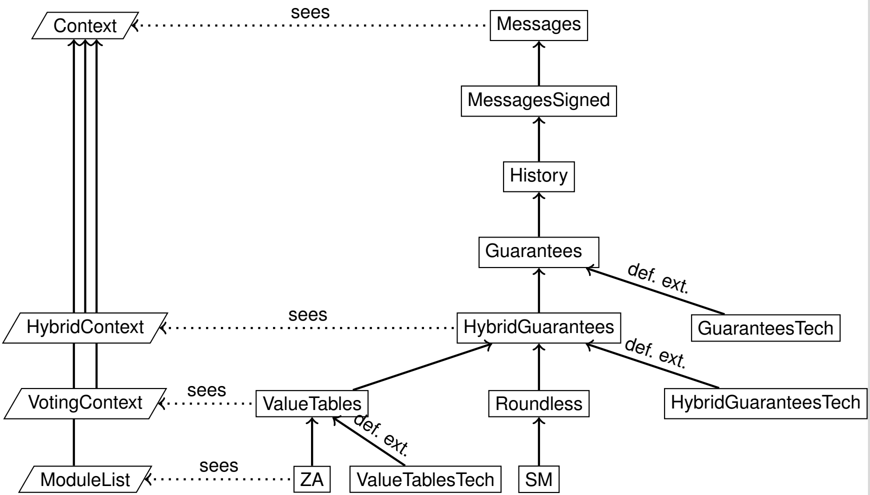
`ROUND_SWITCH refines ROUND`

reproduces the effect of a round change from the internal data.

An implementation would refine `PROCESS_EVENT`.



Refinement Tower



Agreement!

In machine Guarantees:

$$round \geq card(faulty) + 1 \implies \\ (\forall n, m \cdot n \notin faulty \land m \notin faulty \Rightarrow \\ collected(n) = collected(m))$$

Agreement!

In machine Guarantees:

$$\begin{aligned} &\mathit{round} \geq \mathit{card}(\mathit{faulty}) + 1 \implies \\ &\qquad (\forall n, m \cdot n \notin \mathit{faulty} \land m \notin \mathit{faulty} \Rightarrow \\ &\qquad\qquad \mathit{collected}(n) = \mathit{collected}(m)) \end{aligned}$$

In machine HybridGuarantees:

$$\begin{aligned} &\mathit{round} \geq \mathit{card}(\mathit{arbFaulty}) + 1 \implies \\ &\qquad (\forall n, m \cdot n \notin \mathit{faulty} \land m \notin \mathit{faulty} \Rightarrow \\ &\qquad\qquad \mathit{collected}(n) = \mathit{collected}(m)) \end{aligned}$$

Verification Effort

Numbers

Size:	4 contexts, 12 machines, 106 invariants
Labour:	approx. 4 person months
Proofs:	322 proof obligations
Automation:	74/322, 23%