Applications of Formal Verification

Formal Software Design: Modelling in Event-B

Bernhard Beckert · Mattias Ulbrich | SS 2019
Literatur

Jean-Raymond Abrial: Modelling in Event-B
System and Software Engineering
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

Goal: Detect faults here!

Reasons for system faults

- Systems are inherently complex
- Unconsidered situations, corner cases
- Ambiguous natural language requirements
- Component interplay
- ...
The only tool to master complexity is abstraction.

Cliff Jones
Abstraction and Refinement

Abstraction

Abstract

Refinement

Concrete

Abstraction

Beckert, Ulbrich – Applications of Formal Verification
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

**Aspect Behaviour**

refined to

**Aspect Geometry**

refined to
Schematic Diagram vs. PCB Layout

Aspect
“Behaviour” preserved
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.

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**
- 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)

That’s what we will look into in the next sections.
Event-B – Introduction
Event-B 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
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.
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)
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

SETS
  STUDENT // see requirement R1
  SUBJECT

CONSTANTS
  maths  physics  siblings

AXIOMS
  maths ∈ SUBJECT // type of variables
  physics ∈ SUBJECT
  maths ≠ physics // constants could have same value
  siblings ⊆ STUDENT × STUDENT // function type
  ∀s · s ∈ STUDENT ⇒ (s ↦ s) ∉ siblings // irreflexive
  // . . .
MACHINE ExamAbstract
SEES ExamCtxt

VARIABLES
  passed  failed

INVENTRANTS
  passed ⊆ STUDENT  failed ⊆ STUDENT
  passed ∩ failed = ∅  // R3

EVENTS
  INITIALISATION ≜ . . .
  ATTEMPT Exam ≜ . . .  // R2
  PASS Exam ≜ . . .  // R3
  FAIL Exam ≜ . . .  // R3
MACHINE ExamAbstract
VARIABLES passed failed 

EVENTS
INITIALISATION \( \triangleq \)
\[
\text{failed} := \emptyset \\
\text{passed} := \emptyset
\]

PASS\text{EXAM} \( \triangleq \)
\[
\text{ANY } s \text{ grade} \\
\text{WHERE } s \in \text{STUDENT} \wedge \text{grade} \leq 4 \\
\text{THEN } \text{passed} := \text{passed} \cup \{s\}
\]

FAIL\text{EXAM} \( \triangleq \)
\[
\text{ANY } s \text{ grade} \\
\text{WHERE } s \in \text{STUDENT} \wedge \text{grade} > 4 \\
\text{THEN } \text{failed} := \text{failed} \cup \{s\}
\]
MACHINE ExamAbstract
VARIABLES passed failed
INVARINANTS passed ∩ failed = ∅ ... 

EVENTS
PASS_EXAM ≜
  ANY s grade
  WHERE s ∈ STUDENT \ (failed ∪ passed) ∧ grade ≤ 4
  THEN passed := passed ∪ \{s\}

FAIL_EXAM ≜
  ANY s grade
  WHERE s ∈ STUDENT \ (failed ∪ passed) ∧ grade > 4
  THEN failed := failed ∪ \{s\}
Underspecified model

EVENTS
PASS \text{ EXAM} \triangleq
\quad \text{ANY } s \text{ grade WHERE } \text{grade} \leq 4 \land s \in \ldots
\quad \text{THEN } \text{passed} \; := \; \text{passed} \cup \{s\}

FAIL \text{ EXAM} \triangleq
\quad \text{ANY } s \text{ grade WHERE } \text{grade} > 4 \land s \in \ldots
\quad \text{THEN } \text{failed} \; := \; \text{failed} \cup \{s\}

\text{ATTEMPT \text{ EXAM}} \triangleq
\quad \text{ANY } s \text{ grade WHERE } \text{grade} \in \mathbb{N} \land s \in \ldots
\quad \text{THEN } \text{skip}

Additional requirement

\textbf{R4} Any student may attempt the exam three times and ultimately fails if the fourth attempt is unsuccessful.
MACHINE RefinedExams REFINES ExamsAbstract

VARIABLES passed attempts

ININVARIANTS

\[ \text{attempts} \in \text{STUDENT} \rightarrow \mathbb{N} \] // typing for attempts

\[ \text{failed} = \{ s \cdot \text{attempts}(s) = 4 \} \] // coupling invariant

EVENTS

INITIALISATION \( \triangleq \) REFINES INITIALISATION

\[ \text{passed} := \emptyset \]

\[ \text{attempts} := \{ s \cdot s \in \text{STUDENT} \mid (s \mapsto 0) \} \]

EXAMULTIMATEFAIL \( \triangleq \) REFINES EXAMFAIL...

EXAMMISSED \( \triangleq \) REFINES EXAMATTEMPT...

EXAMPASSED \( \triangleq \) REFINES EXAMPASSED...
Refinement Exams (2)

\[
\begin{align*}
\text{EVENTS} \\
\text{ ExamUltimateFail } &\Rightarrow \text{ Refines ExamFail} \\
\text{ Any s grade} &\text{ WHERE } \ldots \land grade > 4 \land \text{ attempts}(s) = 3 \\
\text{ THEN} &\text{ attempts}(s) := \text{ attempts}(s) + 1
\end{align*}
\]

\[
\begin{align*}
\text{ ExamMissed } &\Rightarrow \text{ Refines ExamAttempt} \\
\text{ Any s grade} &\text{ WHERE } \ldots \land grade > 4 \land \text{ attempts}(s) < 3 \\
\text{ THEN} &\text{ attempts}(s) := \text{ attempts}(s) + 1
\end{align*}
\]
Refinement Exams (3)

This refinement takes now also R4 into account.

Refinement preserves invariants

![ Every possible event of RefinedExams is a possible event in ExamsAbstract ]

⇒ 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
- …
Every term in Event-B has a unique type.

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

\[ \text{maths} \in \text{failed} \] is syntactically invalid.

(remember: \( \text{math} \in \text{SUBJECT}, \text{failed} \subseteq \text{STUDENT} \))

“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

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 **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 product).

Every expression $E$ has a unique type $\tau(E)$. 
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 \( \phi \) denotes a set. Let \( R := \{ s \mid s \notin s \} \). Not allowed with types.
One observes: \[ R \in R \iff R \notin R \]
(This crushed naive set theory in early 1900s.)
Sets

Constructors for sets:

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

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

- All relations $E_1 \leftrightarrow E_2$

- All surjections $E_1 \leftrightarrow E_2$  \hspace{1cm} (ran($R$) = $E_2$)

- All total relations $E_1 \leftrightarrow E_2$  \hspace{1cm} (dom($R$) = $E_1$)

- All total surjections $E_1 \leftrightarrow E_2$
Functional relations

Observation

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

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

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

- Total functions $E_1 \rightarrow E_2$
  $$E_1 \rightarrow E_2 = (E_1 \mapsto E_2) \cap (E_1 \leftrightarrow E_2)$$
  (both partial function and total relation)

- Injections $E_1 \leftrightarrow E_2$
  $$(*) \land (\forall x, y, z \cdot x \mapsto z \in R \land y \mapsto z \in R \Rightarrow x = y)$$
Functional relations (2)

Intersection of relation classes give new classes:

- Total injections $E_1 \leftrightarrow E_2 = (E_1 \rightarrow E_2) \cap (E_1 \leftrightarrow E_2)$
- Partial surjections $E_1 \mapsto E_2 = (E_1 \mapsto \rightarrow E_2) \cap (E_1 \leftrightarrow \mapsto E_2)$
- Total surjections $E_1 \rightarrow E_2 = (E_1 \rightarrow E_2) \cap (E_1 \rightarrow \mapsto E_2)$
- Bijections $E_1 \leftrightarrow E_2 = (E_1 \rightarrow E_2) \cap (E_1 \leftrightarrow E_2)$
Example: File system

CONTEXT FileSystemCtx
SETS OBJECT
CONSTANTS files, dirs, root
AXIOMS files ⊆ OBJECT, dirs ⊆ OBJECT,
   root ∈ dirs, files ∩ dirs = ∅

MACHINE FileSystem SEES FileSystemCtx
VARIABLES tree, parent
INVARIANTS
   tree ∈ dirs ↔ (files ∪ dirs)
   // most directories (but root) have a parent directory:
   parent ∈ dirs → dirs
   // more precise
   parent ∈ (dirs \ {root}) → dirs
Relational operations

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

- **Functional application** \(\cdot(\cdot) : \mathcal{P}(S \times T) \times S \to T\)
  \[ x = f(e) \iff e \mapsto x \in f \quad \{ f(e) \} = f[\{e\}] \]

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

**Solution:** Well-definedness needs to be proved

1. \(f \in S \leftrightarrow T\) (not an arbitrary relation in \(S \leftrightarrow T\))
2. \(e \in \text{dom}(f)\)

Everytime a functional application is used.
Restrictions

Concept

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

\[ A \triangleleft R := \{ x, y \cdot x \mapsto y \in R \land x \in A \mid x \mapsto y \} \subseteq R \]

\[ A \triangleleft R := \{ x, y \cdot x \mapsto y \in R \land x \notin A \mid x \mapsto y \} \subseteq R \]

\[ R \triangleright B := \{ x, y \cdot x \mapsto y \in R \land y \in B \mid x \mapsto y \} \subseteq R \]

\[ R \triangleright B := \{ x, y \cdot x \mapsto y \in R \land y \notin B \mid x \mapsto y \} \subseteq R \]

Relational application: \( R[A] = \text{ran}(A \triangleleft R) \)
Override

\[ R \leftarrow S := ((\text{dom } S) \leftarrow R) \cup S \]

\[ x \mapsto y \in R \leftarrow S \iff \begin{cases} x \mapsto y \in S & \text{if } x \in \text{dom}(S) \\ x \mapsto y \in R & \text{if } x \notin \text{dom}(S) \end{cases} \]

- “Clear” \( \text{dom}(S) \) in \( R \) and “replace” by \( S \).
- Special case: \( f \in A \rightarrow B, \ g \in A \rightarrow B \) implies \( f \leftarrow g \in A \rightarrow B \)
- \( f \leftarrow \{x \mapsto y\} \) updates function \( f \) in one place \( x \)

Caution: \( \leftarrow \) and \( \leftarrow \) are different symbols

Syntax sometimes \( \oplus \) instead of \( \leftarrow \)

Compare *Updates* in Dynamic Logic for KeY.
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 \text{files} \subseteq \text{OBJECT}, \text{dirs} \subseteq \text{OBJECT}, \root \in \text{dirs}, \text{files} \cap \text{dirs} = \emptyset

MACHINE FileSystem SEES FileSystemCtx
VARIABLES tree, depth
INVARINTANS
\begin{align*}
\text{tree} \in \text{dirs} & \leftrightarrow (\text{files} \cup \text{dirs}) \land \text{depth} \in \text{dirs} \rightarrow \mathbb{N} \land \\
\forall d \cdot ( (\text{depth}(d) > 0 \Rightarrow \text{depth}[\text{tree}[\{d\}]] = \{\text{depth}(d) - 1\}) \\
& \land (\text{depth}(d) = 0 \Rightarrow \{d\} \triangleleft \text{tree} \triangleright \text{files} = \emptyset))
\end{align*}
Event-B – Events
Machine (systematic)

```plaintext
MACHINE name
SEES context
VARIABLES vars
ININVARIANTS inv(vars)
EVENTS ...
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 $prms$

// the preconditions, conditions on the input values
WHERE $guard(vars, prms)$

// evolution of the program variables when the event “fires”
THEN
  $actions$
END

There is one more construct (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 $t$

### 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
- \( \varphi \) is called the before-after-predicate (BAP)
- Variables without tick: before-state
- Variables with tick: after-state.

Example:

\[ x, y : \quad x' = y' \land y' > y \]

*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

\[ \text{bap}(vars, vars', prms) \]

with

1. \( vars \) the machines variables before the action
2. \( vars' \) the machine variables after the action
3. \( prms \) the parameters of the event

- \( x := t \) is short for \( x :| x' = t \)
- \( x :\in S \) is short for \( x :| x' \in S \)
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”.
- After the first state, only normal events trigger.
Machine Semantics

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

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 $\sigma_0$ is result of the initialisation event
- every state $\sigma_i$ results from an event which operates on $\sigma_{i-1}$ (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. :∈, :|)
- 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: $\overline{T} = T_1 \times \ldots \times T_k$

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

with

- $\exists \text{prms}_{\text{init}} \cdot \text{grd}_{\text{init}}(\text{prms}_{\text{init}}) \land \text{bap}_{\text{init}}(t(0), \text{prms}_{\text{init}})$
- For $n \in \mathbb{N}_1$, there is $e \in \text{EVENTS}$ such that
  $\exists \text{prms}_e \cdot \text{grd}_e(t(i - 1), \text{prms}_e) \land \text{bap}_e(t(i - 1), t(i), \text{prms}_e)$

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

Deadlock: no event $e$ can be triggered, i.e.

$\forall \text{prms}_e \cdot \lnot \text{grd}_e(t(n), \text{prms}_e)$ for all events $e$. 
**SAFETY**: Do all states reachable by $M$ satisfy $inv$?

The red trace violates the invariant in two states.
To show that $\text{inv}(vars)$ is an invariant for machine $M$, one proves for every event:

- Invariants
- Guards of the event
- Before-after-predicate of the event

$\Rightarrow$

modified invariant
Proof Obligation INV

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

1. \( \forall \overline{prms}, \overline{vars}'. \overline{grd}_{init}(\overline{prms}) \land bap_{init}(\overline{vars}', \overline{prms}) \rightarrow inv(\overline{vars}') \)
   (Invariant initially valid)

2. \( \forall \overline{prms}, \overline{vars}, \overline{vars}'. \overline{inv}(\overline{vars}) \land \overline{grd}_e(\overline{vars}, \overline{prms}) \land \overline{bap}_e(\overline{vars}, \overline{vars}', \overline{prms}) \rightarrow inv(\overline{vars}') \)
   for every event \( e \) in \( M \).
   (Events preserve invariant)

Note: Proof Obligation INV is a sufficient criterion, but not necessary. Necessary for inductive invariants.
MACHINE \textit{IndInv}

VARIABLES $x$

INvariants $x \in \mathbb{Z}$ \quad $x \geq 0$

EVENTS

INITIALISATION $\triangleq$

\[ x := 2 \]

STEP $\triangleq$

\[ x := 2 \ast (x - 1) \]

There is only one trace:

\[(2, 2, 2, 2, \ldots)\]

invariant is fulfilled.
Inductive Invariant – Won’t prove

Proof obligation INV for event STEP

\[ \text{inv}(x) \land \text{grd}(x) \land \text{bap}(x, x') \rightarrow \text{inv}(x') \]
\[ x \geq 0 \land x' = 2 \times (x - 1) \rightarrow x' \geq 0 \]

\[ \Leftarrow \text{This is not valid! Invariant is not inductive.} \]

Counter-example: \( x = 0, x' = -2 \)
Show that every action is feasible if the guard is true:

\[
\begin{align*}
\text{Invariants} \\
\text{Guards of the event} \\
\Rightarrow \\
\exists v' \cdot \text{before-after-predicate}
\end{align*}
\]
The action of an event is possible if guard is true.

\[ \forall \text{vars}, \text{prms} \cdot \text{grd}_e(\text{vars}, \text{prms}) \rightarrow \exists \text{vars}' \cdot \text{bap}(\text{vars}, \text{vars}', \text{prms}) \]

Deterministic action: \( x := t \)

...nothing to show

Indeterministic action: \( x \in S \)

...show that \( S \neq \emptyset \)

Indeterministic action: \( x :| \varphi \)

...show satisfiability of \( \varphi \)

Thus impossible evolutions like \( x :| \text{false} \) or \( x \in \emptyset \) are avoided
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:

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

Again, this is sufficient not necessary.
(The invariant may be too weak to imply deadlock freedom)
Event-B – Refinement
Refinement in Event-B

**MACHINE** Abstract
**VARIABLES** \( x \)
**INVARIANTS** \( x \geq 0 \)
**EVENTS** INCREASE \( \equiv \)
\[
(x : | x' \geq x)
\]

**MACHINE** Refined
**REFINES** Abstract
**VARIABLES** \( x \)
**EVENTS** NEXT \( \text{VAL} \) \( \equiv \)
**REFINES** INCREASE
\[
x := 5 \times x^2 + 3 \times x
\]
Refining Machines

MACHINE Abstract

SEES Context

VARIABLES \( vars_A \)

INVARIANTS

\[ inv_A(vars_A) \]

EVENTS

INITIALISATION \( \equiv \ldots \)

EVT_A \( \equiv \ldots \)

END

MACHINE Refined

REFINES Abstract

SEES Context

VARIABLES \( vars_R \)

INVARIANTS

\[ inv_R(vars_A, vars_R) \]

EVENTS

INITIALISATION \( \equiv \ldots \)

EVT_R \( \equiv \)

REFINES EVT_A \ldots

END
Machines as Relations

Every machine $M$ defines:

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

Details

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

$$I_M(p) = \{ s \in S_M \mid \text{grd}_{\text{init}}(p) \land \text{bap}_{\text{init}}(s', p) \}$$

$$I_M = \bigcup_{p} I_M(p)$$

$$E_{M;\text{evt}}(p) = \{ (s \mapsto s') \mid \text{grd}_{\text{evt}}(s, p) \land \text{bap}_{\text{evt}}(s, s', p) \}$$

$$E_{M;\text{evt}} = \bigcup_{p} E_{M;\text{evt}}(p)$$
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. $E_{R;\text{evt}_R} \subseteq E_{A;\text{evt}_A}$ for each event

($\text{evt}_R$ is the event in $R$ that refines event $\text{evt}_A$ from $A$)
Loss of behaviour

Why is this problematic?

MACHINE A ...  
EVENT emergencyStop  \equiv  
WHERE true THEN heavyMachine := stop  
END

refined by

MACHINE R ...  
EVENT emergencyStop  \equiv \text{REFINES} emergencyStop  
WHERE false THEN heavyMachine := stop  
END

\[ E_{R;\text{evt}} = \emptyset \implies \text{R refines A} \]
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. $I_R \neq \emptyset$
3. $E_{R;evt_R} \subseteq E_{A;evt_A}$ for each event
4. $\text{dom}(E_{A;evt_A}) \subseteq \text{dom}(E_{R;evt_R})$ for each event
Coupling

More general notion of refinement

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

→ **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*

$C = \{ r \mapsto a \mid a = \text{dom}(r) \} = \{ f, m \cdot (f \mapsto m) \mapsto m \}$
Refinement – Coupling

- Sensible to assume \( C \) a total relation:
  \[
  C \in S_R \leftrightarrow S_A
  \]

- 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: $R$ refines $A$

For every concrete trace $(\chi_0, \chi_1, \ldots)$ of $R$ with events $(\text{evt}_1^R, \text{evt}_2^R, \ldots)$ there exists an abstract trace $(\sigma_0, \sigma_1, \ldots)$ with events $(\text{evt}_1^A, \text{evt}_2^A, \ldots)$ such that

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

1. $I_R \subseteq C^{-1}[I_A]$ and
2. $E_{R;\text{evt}_R} \subseteq C ; E_{A;\text{evt}_A} ; C^{-1}$ for each event.

$$(\forall x, y \cdot x \leftrightarrow y \in R^{-1} \iff y \not\leftrightarrow x \in R, \text{ inverse relation})$$
Refinement – Path subsumption

\[ \sigma_n \xrightarrow{\text{evt}_A} \sigma_{n+1} \]

\[ C \subseteq E_R; \text{evt}_R \subseteq C; E_A; \text{evt}_A; C^{-1} \]
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)**

<table>
<thead>
<tr>
<th>MACHINE</th>
<th>VARIABLES</th>
<th>INVARIANTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractFileSys</td>
<td>openFiles</td>
<td>openFiles $\subseteq$ FILES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MACHINE</th>
<th>VARIABLES</th>
<th>INVARIANTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefinedFileSys</td>
<td>openModes</td>
<td>openModes $\subseteq$ FILES $\times$ MODES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>openFiles $=$ dom(openModes)</td>
</tr>
</tbody>
</table>
Proof Obligation GRD

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

\[ \Rightarrow \text{Abstraction is enabled when refinement is.} \]

Abstract invariants
Concrete invariants
Concrete event guard

\[ \Rightarrow \text{Abstract event guard} \]

\[ \forall vars_A, vars_R \cdot \]
\[ inv_A(vars_A) \land inv_R(vars_A, vars_R) \land grd_R(vars_R) \]
\[ \Rightarrow grd_A(vars_A) \]

(Version w/o parameters, see literature for full version)
Proof Obligation \textbf{SIM}

Show that refined action \textit{simulates} abstract actions

Abstract invariants
Concrete invariants
Concrete event guard
Concrete before-after-predicate

\[ \Rightarrow \]
Abstract before-after-predicate

\textbf{Rem} \quad E_{R;\text{evt}_R} \subseteq C ; E_{A;\text{evt}_A} ; C^{-1}

\textbf{Obs} \quad \text{The coupling invariant is only used for the before-state not for the after-state.}

? Why?

! Already proven condition \textit{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
- ...
Byzantine Agreement –
A case study verified with Event-B

Based on:
Byzantine Generals

“When shall we attack?”

agree on a time even in the presence of traitors
Application in Avionics

“Which components are operative?”

agree on the set of operative components even in the presence of faulty components
Example Run 2

Round 2

C1

C2

C3

C4
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.

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

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

**SETS**
- MODULE
- VALUE

**CONSTANTS**
- faulty, transmitter, $V_0$

**AXIOMS**
- $\text{faulty} \subseteq \text{MODULE}$
- $\text{transmitter} \in \text{MODULE}$
- $V_0 \in \text{VALUE}$
- $\text{finite}(\text{faulty})$

**END**
MACHINE Messages
SEES Context

VARIABLES messages, round, collected

INVARINTS

- ty_mess : messages ⊆ Module × Module × Value
- ty_round : round ∈ N
- ty_collected : collected ∈ Module → ℙ(Value)

...
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 $\cong \ldots$

EVENT ROUND $\cong$

$\text{act}1 : \text{round} := \text{round} + 1$
$\text{act}2 : \text{messages} \in \mathcal{P}(\text{MODULE} \setminus \{\text{transmitter}\} \times \text{MODULE} \times \text{VALUE})$
$\text{act}3 : \text{collected} := \lambda m \cdot \text{collected}(m) \cup \{v \mid (s, m, v) \in \text{messages}\}$

END
First refinement: signed messages

All messages are signed in a trustworthy manner:
No forgery possible $\implies$ Consider only \textit{relayed} messages.

round $k$:
\[ s \xrightarrow{\vee} r \]

round $k + 1$:
\[ r \xrightarrow{\vee} n \]
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
val1: $\forall s, r, v \cdot (s, r, v) \in \text{messages} \Rightarrow v \in \text{collected(\text{transmitter})}$
val2: $\forall n \cdot \text{collected}(n) \subseteq \text{collected(\text{transmitter})}$

Events
EVENT ROUND REFINES ROUND $\triangleq$
act1, act3 as above
act2: $\text{messages} : \in \mathcal{P}(\{(r, n, v) \mid (s, r, v) \in \text{messages}\})$
was: $\text{messages} : \in \mathcal{P}((\text{MODULE} \setminus \{\text{transmitter}\} \times \text{MODULE} \times \text{VALUE}))$

End
Refinement Tower

covered so far

Context

Messages

MessagesSigned

History

Guarantees

GuaranteesTech

HybridGuarantees

HybridGuaranteesTech

Roundless

SM

ValueTablesTech

ValueTables

ZA

ModuleList

VotingContext

HybridContext

Beckert, Ulbrich – Applications of Formal Verification
Agreement!

In machine Guarantees:

\[ \text{round} \geq \text{card}(\text{faulty}) + 1 \implies \]
\[ (\forall n, m \cdot n \notin \text{faulty} \land m \notin \text{faulty} \implies \]
\[ \text{collected}(n) = \text{collected}(m)) \]

In machine HybridGuarantees:

\[ \text{round} \geq \text{card}(\text{arbFaulty}) + 1 \implies \]
\[ (\forall n, m \cdot n \notin \text{faulty} \land m \notin \text{faulty} \implies \]
\[ \text{collected}(n) = \text{collected}(m)) \]
## Verification Effort

### Numbers

<table>
<thead>
<tr>
<th>Size:</th>
<th>4 contexts, 12 machines, 106 invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labour:</td>
<td>approx. 4 person months</td>
</tr>
<tr>
<td>Proofs:</td>
<td>322 proof obligations</td>
</tr>
<tr>
<td>Automation:</td>
<td>74/322, 23%</td>
</tr>
</tbody>
</table>