Designing Safe Cybper-Physical Systems

A proof and refinement based approach

Guillaume Dupont

IRIT, Toulouse INP – ENSEEIHT[1]

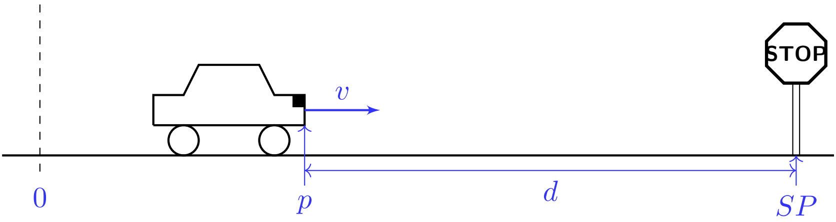
ABZ 2025

[1] This work was supported by grant ANR-17-CE25-0005 (DISCONT Project https://discont.loria.fr)

Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

An example: automatic brake



A car (p, v, a) must stop before SP

$\Rightarrow$ *design a* ***correct controller*** *that stops the car in time*

Problem : controller = *program*, car = *physical object*

$\Rightarrow$ *controller characterised by* ***code***

$\Rightarrow$ *car characterised by* ***differential equations***

Hybrid systems

Definition

Hybrid systems (HS) integrate both **discrete** and **continuous** behaviours.

Hybrid systems

Definition

Hybrid systems (HS) integrate both **discrete** and **continuous** behaviours.

$\Rightarrow$ ***hybrid*** *nature, makes reasoning difficult*

Hybrid systems

Definition

Hybrid systems (HS) integrate both **discrete** and **continuous** behaviours.

$\Rightarrow$ ***hybrid*** *nature, makes reasoning difficult*

We want a formal method for modelling and verifying HS $\Rightarrow$ *integration of discrete and continuous aspects* ***at the same level***

Hybrid systems

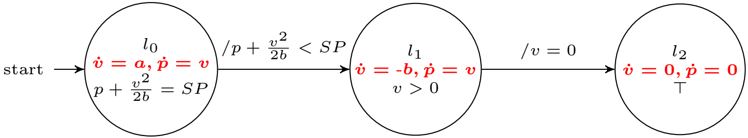
Definition

Hybrid systems (HS) integrate both **discrete** and **continuous** behaviours.

$\Rightarrow$ ***hybrid*** *nature, makes reasoning difficult*

We want a formal method for modelling and verifying HS $\Rightarrow$ *integration of discrete and continuous aspects* ***at the same level***

Ex.: **hybrid automata** [Alu+95] : automata + continuous



Hybrid systems

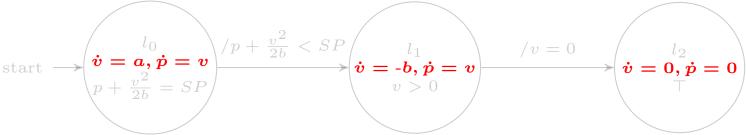
Definition

Hybrid systems (HS) integrate both **discrete** and **continuous** behaviours.

$\Rightarrow$ ***hybrid*** *nature, makes reasoning difficult*

We want a formal method for modelling and verifying HS $\Rightarrow$ *integration of discrete and continuous aspects* ***at the same level***

Ex.: **hybrid automata** [Alu+95] : automata + continuous



Event-B

Definition

Event-B is formal method for the **correct-by-construction** design of complex systems[Abr10].

Event-B

Definition

Event-B is formal method for the **correct-by-construction** design of complex systems[Abr10].

System = **state** (variables) + **events** + **invariants**

Event = **guard** + **state transition** (BAP)

Event-B

Definition

Event-B is formal method for the **correct-by-construction** design of complex systems[Abr10].

System = **state** (variables) + **events** + **invariants**

Event = **guard** + **state transition** (BAP)

```
MACHINE Car
VARIABLES p, v, state
INVARIANTS
 inv1: p ∈ ℤ
 inv2: v ∈ ℤ
 inv3: state ∈ {l0, l1, l2}
 inv4: p < SP
EVENTS
 INITIALISATION
 THEN
  act1: p, v := p0, v0
  act2: state := l0
 END
```

```
move
ANY Δt
WHERE
 grd1: Δt ∈ ℕ ∧ p + v × Δt < SP
THEN
 act1: p :| p′ = p + v × Δt
END

close
WHEN
 grd1: p + (v × v)/(2 × b) = SP
THEN
 act1: state := l1
END
```

```
brake
ANY Δt
WHERE
 grd1: state = l1
 grd2: Δt ∈ ℕ ∧ v − b × Δt > 0
THEN
 act1: v :| v′ = v − b × Δt
END

stop
WHERE
 grd1: v = 0
THEN
 act1: state := l2
END
```

Event-B

Definition

Event-B is formal method for the **correct-by-construction** design of complex systems[Abr10].

System = **state** (variables) + **events** + **invariants**

Event = **guard** + **state transition** (BAP)

```
MACHINE Car
VARIABLES p, v, state
INVARIANTS
 inv1: p ∈ ℤ
 inv2: v ∈ ℤ
 inv3: state ∈ {l0, l1, l2}
 inv4: p < SP
EVENTS
 INITIALISATION
 THEN
  act1: p, v := p0, v0
  act2: state := l0
 END
```

```
move
ANY Δt
WHERE
 grd1: Δt ∈ ℕ ∧ p + v × Δt < SP
THEN
 act1: p :| p′ = p + v × Δt
END

close
WHEN
 grd1: p + (v × v)/(2 × b) = SP
THEN
 act1: state := l1
END
```

```
brake
ANY Δt
WHERE
 grd1: state = l1
 grd2: Δt ∈ ℕ ∧ v − b × Δt > 0
THEN
 act1: v :| v′ = v − b × Δt
END

stop
WHERE
 grd1: v = 0
THEN
 act1: state := l2
END
```

Event-B supports **refinement**: gradual inclusion of details while preserving properties (hence correctness by construction)

Theory extension

Event-B is based on **first order logic** and **set theory**
⇒ *expressive but low-level, lack of reusable higher order constructs*
Solution: the **theory component** [BM13]

Theory = algebraic/axiomatic datatypes + operators and properties

```
THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...
DATATYPES
  Type1(E, ...)
  constructors cstr1(p1: T1, ...), ...
OPERATORS
  Op1 <nature> (p1: T1, ...)
    well-definedness WD(p1,...)
    direct definition D1
```

```
AXIOMATIC DEFINITIONS
  TYPES A1, ...
  OPERATORS
   AOp2 <nature> (p1: T1, ...): Tr
     well-definedness WD(p1,...)
  AXIOMS A1, ...
THEOREMS T1, ...
PROOF RULES
  ...
END
```

⇒ *theories used to formalise **mathematical concepts** (continuous functions, diff. eq.) and **domain knowledge** (trains, cars)*

Outline

1. Introduction
   - Context
   - Event-B and theories

2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation

3. Conclusion and Future Work

Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

Modelling HS: How?

We want in the same model:

- discrete behaviours *[easy!]*
- continuous dynamics: "dense" time, continuous functions, diff. equations

Idea: try to elaborate a general HS schema

Modelling HS: How?

We want in the same model:

- discrete behaviours *[easy!]*
- continuous dynamics: "dense" time, continuous functions, diff. equations

Idea: try to elaborate a general HS schema

Continuous state variables = *functions of time* ($\in \mathbb{R} \nrightarrow S$)
$\Rightarrow$ ***continuous evolution*** *as CBAP*

$$\begin{aligned}
\mathbf{CBAP}(t, t', x_p, x'_p, \mathcal{P}, H) \;\equiv\; x_p \mathbin{:|_{t \to t'}} \mathcal{P}(x_p, x'_p) \,\&\, H \;\equiv& \\
[0, t[\lhd x'_p = [0, t[\lhd x_p & \qquad (\textit{Past Preservation}) \\
\wedge \mathcal{P}([0, t] \lhd x_p, [t, t'] \lhd x'_p) & \qquad (\textit{Predicate}) \\
\wedge \forall t^* \in [t, t'], x_p(t^*) \in H & \qquad (\textit{Evolution Dom.})
\end{aligned}$$

Note: *shorthand for differential equations:*

$$x_p \mathbin{:\sim_{t \to t'}} \mathcal{E} \,\&\, H \;\equiv\; x_p :|_{t \to t'} \mathbf{solutionOf}([t, t'], \mathcal{E}, x'_p) \,\&\, H$$

Continuous Assignment – Properties

CBAP associated to particular proved **meta-theorems**:

Continuous Assignment – Properties

CBAP associated to particular proved **meta-theorems**:

- **Well-Definedness**: assignment is well-defined iff
  1. $t < t'$ (time progression)
  2. $\forall u, v \cdot \mathcal{P}(u, v) \Rightarrow u \in \mathbb{R}^+ \nrightarrow S \land v \in \mathbb{R}^+ \nrightarrow S$
     $\land [0, t[\subseteq \mathrm{dom}(u) \land [t, t'] \subseteq \mathrm{dom}(v)$ (type/domain coherence)

Continuous Assignment – Properties

CBAP associated to particular proved **meta-theorems**:

- **Well-Definedness**: assignment is well-defined iff
  1. $t < t'$ (time progression)
  2. $\forall u, v \cdot \mathcal{P}(u, v) \Rightarrow u \in \mathbb{R}^+ \nrightarrow S \wedge v \in \mathbb{R}^+ \nrightarrow S$
     $\wedge [0, t[\subseteq \mathrm{dom}(u) \wedge [t, t'] \subseteq \mathrm{dom}(v)$ (type/domain coherence)

- **Feasibility**: there exists $x_p^0 \in \mathbb{R} \nrightarrow S$ with $[t, t'] \subseteq \mathrm{dom}(x_p^0)$ s.t.:
  1. $\mathcal{P}([0, t] \lhd x_p, x_p^0)$ (predicate holds)
  2. $\forall t^* \in [t, t'], x_p^0(t^*) \in H$ (evolution domain holds)

  $\Rightarrow$ *reachibility of next state* t'

Continuous Assignment – Properties

CBAP associated to particular proved **meta-theorems**:

- **Well-Definedness**: assignment is well-defined iff
  1. $t < t'$ (time progression)
  2. $\forall u, v \cdot \mathcal{P}(u, v) \Rightarrow u \in \mathbb{R}^+ \nrightarrow S \wedge v \in \mathbb{R}^+ \nrightarrow S$
     $\wedge [0, t[\subseteq \mathrm{dom}(u) \wedge [t, t'] \subseteq \mathrm{dom}(v)$ (type/domain coherence)

- **Feasibility**: there exists $x_p^0 \in \mathbb{R} \nrightarrow S$ with $[t, t'] \subseteq \mathrm{dom}(x_p^0)$ s.t.:
  1. $\mathcal{P}([0, t] \lhd x_p, x_p^0)$ (predicate holds)
  2. $\forall t^* \in [t, t'], x_p^0(t^*) \in H$ (evolution domain holds)

  $\Rightarrow$ *reachibility of next state* t'

- **Invariant preservation** (continuous induction): for establishing invariant $\mathcal{I} \subseteq S$ on $[0, t']$, it is sufficient that:
  1. $\forall t^* \in [0, t[, x_p(t^*) \in \mathcal{I}$
  2. $\mathbf{CBAP}(t, t', x_p, x_p', \mathcal{P}, H \cap \mathcal{I})$

Continuous Assignment – Properties

CBAP associated to particular proved **meta-theorems**:

- **Well-Definedness**: assignment is well-defined iff
  1. $t < t'$ (time progression)
  2. $\forall u, v \cdot \mathcal{P}(u, v) \Rightarrow u \in \mathbb{R}^+ \nrightarrow S \wedge v \in \mathbb{R}^+ \nrightarrow S$ $\wedge [0, t[\subseteq \mathrm{dom}(u) \wedge [t, t'] \subseteq \mathrm{dom}(v)$ (type/domain coherence)

- **Feasibility**: there exists $x_p^0 \in \mathbb{R} \nrightarrow S$ with $[t, t'] \subseteq \mathrm{dom}(x_p^0)$ s.t.:
  1. $\mathcal{P}([0, t] \lhd x_p, x_p^0)$ (predicate holds)
  2. $\forall t^* \in [t, t'], x_p^0(t^*) \in H$ (evolution domain holds)

  $\Rightarrow$ *reachibility of next state t'*

- **Invariant preservation** (continuous induction): for establishing invariant $\mathcal{I} \subseteq S$ on $[0, t']$, it is sufficient that:
  1. $\forall t^* \in [0, t[, x_p(t^*) \in \mathcal{I}$
  2. $\mathbf{CBAP}(t, t', x_p, x_p', \mathcal{P}, H \cap \mathcal{I})$

$\Rightarrow$ ***instantiated** to discharge POs for continuous events*

Modelling Features

```
THEORY DiffEq IMPORT Functions
TYPE PARAMETERS E, F
DATATYPES
 DE(F) constructors ode(f, η0, t0), ...
OPERATORS
 solutionOf predicate (D : ℙ(ℝ), η : ℝ ⇸ F, ℰ : DE(F)) ...
 Solvable predicate (D : ℙ(ℝ), ℰ : DE(F)) ...
 CBAP predicate (t, t' : ℝ+, x_p, x'_p : ℝ ⇸ F, 𝒫 : ℙ((ℝ ⇸ F) × (ℝ ⇸ F)), H : ℙ(F)) ...
 :∼ predicate (t, t' : ℝ+, x_p, x'_p : ℝ ⇸ F, ℰ : DE(F), H : ℙ(F))
   well-definedness condition  Solvable([t, t'], ℰ)
   direct definition  solutionOf([t, t'], x'_p, ℰ) ∧ ...
 ...
AXIOMS
 CauchyLipschitz: -- external
   ∀ℰ, D, D_F · ℰ ∈ DE(F) ∧ ... ⇒ Solvable(D, ℰ)
 ...
THEOREMS
 CBAPINV:
   ∀t, t', η, η', 𝒫, H, ℐ · t, t' ∈ ℝ ∧ η, η' ∈ ℝ ⇸ F ∧ ... ⇒ (∀t* · t* ∈ [0, t'] ⇒ η'(t*) ∈ ℐ)
 ...
```

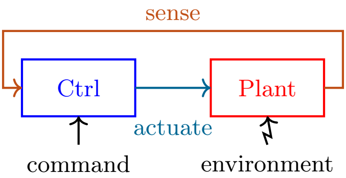
- use of theories to integrate continuous features
  $\Rightarrow$ *e.g. continuous behaviour using differential equations*
- exploit WD to ensure correct use of operators/theorems

Modelling Features

THEORY DiffEq IMPORT Functions
TYPE PARAMETERS E, F
DATATYPES
 $\mathbf{DE}(F)$ constructors $\mathbf{ode}(f, \eta_0, t_0)$, ...
OPERATORS
 solutionOf *predicate* $(D : \mathbb{P}(\mathbb{R}),\ \eta : \mathbb{R} \nrightarrow F,\ \mathcal{E} : \mathbf{DE}(F))$ ...
 Solvable *predicate* $(D : \mathbb{P}(\mathbb{R}),\ \mathcal{E} : \mathbf{DE}(F))$ ...
 CBAP *predicate* $(t, t' : \mathbb{R}^+,\ x_p, x'_p : \mathbb{R} \nrightarrow F,\ \mathcal{P} : \mathbb{P}((\mathbb{R} \nrightarrow F) \times (\mathbb{R} \nrightarrow F)),\ H : \mathbb{P}(F))$ ...
 :∼ *predicate* $(t, t' : \mathbb{R}^+,\ x_p, x'_p : \mathbb{R} \nrightarrow F,\ \mathcal{E} : \mathbf{DE}(F),\ H : \mathbb{P}(F))$
  well-definedness condition $\mathbf{Solvable}([t, t'], \mathcal{E})$
  direct definition $\mathbf{solutionOf}([t, t'], x'_p, \mathcal{E}) \land \ldots$
 ...
AXIOMS
 CauchyLipschitz: -- *external*
  $\forall \mathcal{E}, D, D_F \cdot \mathcal{E} \in \mathbf{DE}(F) \land \ldots \Rightarrow \mathbf{Solvable}(D, \mathcal{E})$
 ...
THEOREMS
 CBAPINV:
  $\forall t, t', \eta, \eta', \mathcal{P}, H, \mathcal{I} \cdot t, t' \in \mathbb{R} \land \eta, \eta' \in \mathbb{R} \nrightarrow F \land \ldots \Rightarrow (\forall t^* \cdot t^* \in [0, t'] \Rightarrow \eta'(t^*) \in \mathcal{I})$
 ...

- use of theories to integrate continuous features
  $\Rightarrow$ *e.g. continuous behaviour using differential equations*
- exploit WD to ensure correct use of operators/theorems

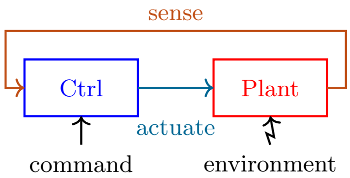
Event-B "Hybridation"



Generic schema for HS

- discrete controller (program)
- continuous "plant" (physical)
- sensing and actuation events

Event-B "Hybridation"



Generic schema for HS

- discrete controller (program)
- continuous "plant" (physical)
- sensing and actuation events

- *Dense* time $t \in \mathbb{R}$
- Discrete: discrete variables x_s + BAP
- Continuous: continues continuous x_p + CBAP

MACHINE Generic
VARIABLES t, x_s, x_p
INVARIANTS
  inv1: $t \in \mathbb{R}^+$
  inv2: $x_s \in STATES$
  inv3: $x_p \in \mathbb{R} \rightarrow S$
  inv4: $[0, t] \subseteq \mathrm{dom}(x_p)$

Generic Model (Cont'd)

- **Event parameters** for genericity
- Sensing with guard on continuous state and discrete state (`grd3`)

```
Sense
ANY s, p
WHERE
  grd1: s ∈ ℙ1(STATES)
  grd2: p ∈ ℙ(STATES × ℝ × S)
  grd3: (x_s ↦ t ↦ x_p(t)) ∈ p
THEN
  act1: x_s :∈ s
END
```

Generic Model (Cont'd)

- **Event parameters** for genericity
- Sensing with guard on continuous state and discrete state (`grd3`)

```
Sense
ANY s, p
WHERE
  grd1: s ∈ ℙ1(STATES)
  grd2: p ∈ ℙ(STATES × ℝ × S)
  grd3: (x_s ↦ t ↦ x_p(t)) ∈ p
THEN
  act1: x_s :∈ s
END
```

```
Actuate
ANY 𝒫, s, H, t'
WHERE
  grd0: t' > t
  grd1: 𝒫 ∈ (ℝ+ ⇸ S) × (ℝ+ ⇸ S)
  grd2: Feasible([t, t'], x_p, 𝒫, H)
  grd3: s ⊆ STATES
  grd4: x_s ∈ s
  grd5: H ⊆ S
  grd6: x_p(t) ∈ H
THEN
  act1: x_p :|_{t→t'} 𝒫(x_p, x'_p) & H
END
```

- Model plant's behaviour
- Continuous event based on CBAP $\Rightarrow$ *generic continuous behaviour $\mathcal{P}$*
- Feasibility: **Feasible** guard
- Associated discrete state
- Constrained by evolution domain

Example: Stopping Car

```
MACHINE Car REFINES Generic
VARIABLES t, x_s, p, v
INVARIANTS
 inv31-32: p ∈ ℝ ⇸ S, v ∈ ℝ ⇸ S
 inv41-42: [0, t] ⊆ dom(p),[0, t] ⊆ dom(v)
 inv5: x_p = [ v p ]^⊤
 inv6: ∀t* · t^a st ∈ [0, t] ⇒ p(t) ≤ SP
```

```
sense_close REFINES Sense
WHERE grd1: x_s = l_0
      grd2: p(t) + v(t)^2/2 ≥ SP
WITH s : s = {l_1}
     p : p = {p*, v* | p* + v*^2/2 ≥ SP}
THEN act1: x_s := l_1
END
```

```
actuate_move REFINES Actuate
ANY t'
WHERE grd0: t' > t
      grd1: x_s = l_0
      grd2: p(t) + v(t)^2/2 < SP
WITH eq : eq = ode(f_move, [ v(t) p(t) ]^⊤ , t)
     s : s = {l_0}
     x'_p : x'_p = [ v' p' ]^⊤
     H : H = {v*, p* | p* + v*^2/2 ≥ SP}
THEN act1: v, p :∼_{t→t'}
  ode(f_move, [ v(t) p(t) ]^⊤ , t)
  &{v*, p* | p* + v*^2/2 ≥ SP}
```

- **Instantiation = refinement** $\Rightarrow$ ***witnesses*** *(*`WITH`*) and* ***gluing invariant*** *(*`inv5`*) provided*
- Continuous behaviour = **ODE** $\Rightarrow$ *ODE solvability* ***required by WD of*** $:\sim_{t \to t'}$ $\Rightarrow$ *by GS:*

$$\textit{solvability} \Rightarrow \textbf{Feasible}$$

Example: Stopping Car

```
MACHINE Car REFINES Generic
VARIABLES t, x_s, p, v
INVARIANTS
 inv31-32: p ∈ ℝ ⇸ S, v ∈ ℝ ⇸ S
 inv41-42: [0, t] ⊆ dom(p), [0, t] ⊆ dom(v)
 inv5: x_p = [v p]^⊤
 inv6: ∀t* · t^a st ∈ [0, t] ⇒ p(t) ≤ SP
```

```
sense_close REFINES Sense
WHERE grd1: x_s = l_0
      grd2: p(t) + v(t)^2/2 ≥ SP
WITH s : s = {l_1}
     p : p = {p*, v* | p* + v*^2/2 ≥ SP}
THEN act1: x_s := l_1
END
```

```
actuate_move REFINES Actuate
ANY t'
WHERE grd0: t' > t
      grd1: x_s = l_0
      grd2: p(t) + v(t)^2/2 < SP
WITH eq : eq = ode(f_move, [v(t) p(t)]^⊤, t)
     s : s = {l_0}
     x'_p : x'_p = [v' p']^⊤
     H : H = {v*, p* | p* + v*^2/2 ≥ SP}
THEN act1: v, p :∼_{t→t'}
  ode(f_move, [v(t) p(t)]^⊤, t)
  &{v*, p* | p* + v*^2/2 ≥ SP}
```

- **Instantiation = refinement** $\Rightarrow$ ***witnesses*** *(`WITH`) and* ***gluing invariant*** *(`inv5`) provided*
- Continuous behaviour = **ODE** $\Rightarrow$ *ODE solvability* ***required by WD of*** $:\sim_{t \to t'}$ $\Rightarrow$ *by GS:*

$$\textit{solvability} \Rightarrow \textbf{Feasible}$$

Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

Towards a formal framework

Idea:

- use *algebraic theories* to extend Event-B
  $\Rightarrow$ *CBAP, diff. eq. + formal properties*
- define a **parameterised generic model** of hybrid systems
  $\Rightarrow$ *refinement-instantiation to derive any HS*
- parameterised refinement of generic model = applicable to any HS
  $\Rightarrow$ *definition of formal **design patterns***

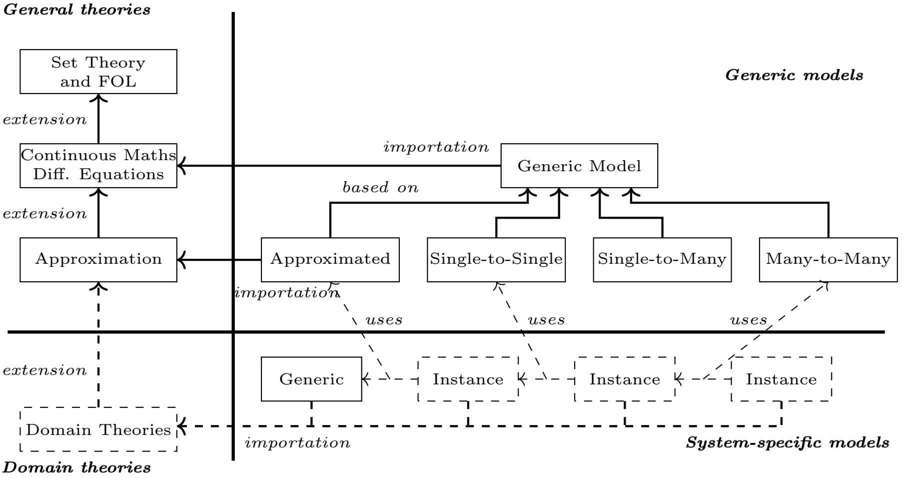
Towards a formal framework

Idea:

- use *algebraic theories* to extend Event-B
  $\Rightarrow$ *CBAP, diff. eq. + formal properties*
- define a **parameterised generic model** of hybrid systems
  $\Rightarrow$ *refinement-instantiation to derive any HS*
- parameterised refinement of generic model = applicable to any HS
  $\Rightarrow$ *definition of formal* ***design patterns***

In a nutshell, designing a hybrid system involves a **refinement chain** stemming from the **generic model** and consisting of **design pattern application**

Framework – Overview



Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

Architectural patterns

Decompose HS in multiple interacting components:

- one controller + one plant *("single-to-single")*
  $\Rightarrow$ *generic model*
- one controller + multiple plants *("single-to-many")*
  $\Rightarrow$ *centralised control of multiple components*
- multiple controllers + multiple plants *("many-to-many")*
  $\Rightarrow$ *distributed HS, cyber-physical system*

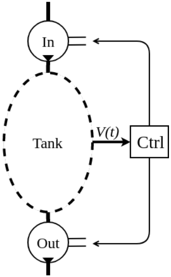
Architectural patterns

Decompose HS in multiple interacting components:

- one controller + one plant *("single-to-single")*
  ⇒ *generic model*
- one controller + multiple plants *("single-to-many")*
  ⇒ *centralised control of multiple components*
- multiple controllers + multiple plants *("many-to-many")*
  ⇒ *distributed HS, cyber-physical system*

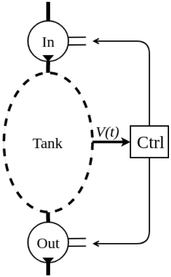
Idea: introduce architecture with a pattern
⇒ *challenge: link global state to local state*

S2M, Centralised control (Example)

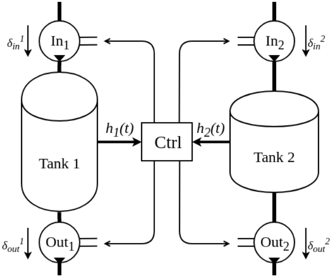


- Abstract tank, volume $V(t)$
- Controller state x_s
  (Filling, Emptying, ...)
  $\Rightarrow$ control In/Out pumps
- Safety: $V_{low} \leq V \leq V_{high}$

S2M, Centralised control (Example)



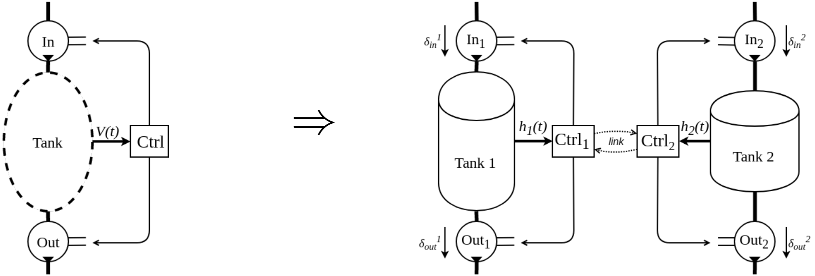
$\Rightarrow$



- Abstract tank, volume $V(t)$
- Controller state x_s
  (Filling, Emptying, ...)
  $\Rightarrow$ control In/Out pumps
- Safety: $V_{low} \leq V \leq V_{high}$

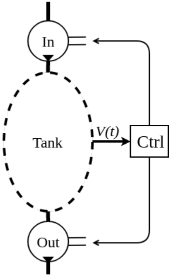
- 2 cylindrical tanks (B_i)
- Sensing **height** (h_i)
  $\Rightarrow V(t) = B_1 h_1(t) + B_2 h_2(t)$
- Centralised controller + policy
  $\Rightarrow P(x_s, In_1, Out_1, In_2, Out_2)$

M2M, Distributed HS (Example)

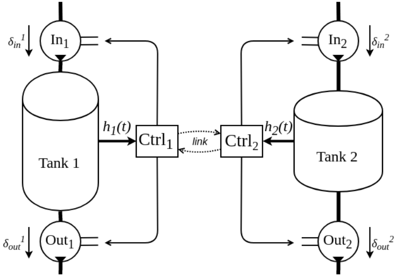


- Same situation but **independant HS** ($x_{s,1}, h_1$ et $x_{s,2}, h_2$)
  $\Rightarrow$ *still* $V(t) = B_1 h_1(t) + B_2 h_2(t)$
  $\Rightarrow$ *policy between discrete states* $P(x_s, x_{s,1}, x_{s,2})$

M2M, Distributed HS (Example)



$\Rightarrow$



- Same situation but **independant HS** ($x_{s,1}, h_1$ et $x_{s,2}, h_2$)
  $\Rightarrow$ *still* $V(t) = B_1 h_1(t) + B_2 h_2(t)$
  $\Rightarrow$ *policy between discrete states* $P(x_s, x_{s,1}, x_{s,2})$
- Imperfect communication = imprecision, **no global state**
  - each component *estimates* the others $\Rightarrow h_i^{sim}$
  - precision Δ^{sim}: $|h_i - h_i^{sim}| \leq \Delta^{sim}$
  - predicate strengthening:
    $V \leq V_{high} \;\rightarrow\; B_1 h_1 + B_2 h_2^{sim} \leq V_{high} - \boldsymbol{\Delta^{sim}}$

Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

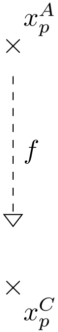
Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Continuous refinement: $x_p^A = f(x_p^C)$



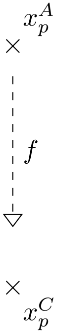
Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Continuous refinement: $x_p^A = f(x_p^C)$

$\Rightarrow$ *Let's "loosen" equality:* $x \overset{\boldsymbol{\delta}}{\approx} y \;\equiv\; d(x, y) \leq \delta$



Approximation pattern

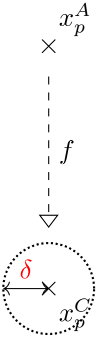
HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Continuous refinement: $x_p^A = f(x_p^C)$

$\Rightarrow$ *Let's "loosen" equality:* $x \overset{\boldsymbol{\delta}}{\approx} y \;\equiv\; d(x,y) \leq \delta$

$\Rightarrow$ Approximate refinement: $x_p^A \overset{\boldsymbol{\delta}}{\approx} f(x_p^C)$



Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

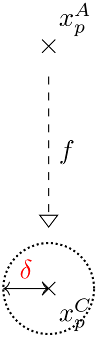
Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Continuous refinement: $x_p^A = f(x_p^C)$

$\Rightarrow$ *Let's "loosen" equality:* $x \overset{\boldsymbol{\delta}}{\approx} y \;\equiv\; d(x,y) \leq \delta$

$\Rightarrow$ Approximate refinement: $x_p^A \overset{\boldsymbol{\delta}}{\approx} f(x_p^C)$

Property: $x_p^A \in S$



Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use **approximation***

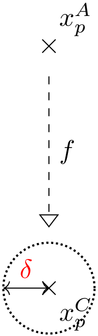
Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a **refinement***

Continuous refinement: $x_p^A = f(x_p^C)$

$\Rightarrow$ *Let's "loosen" equality:* $x \overset{\boldsymbol{\delta}}{\approx} y \;\equiv\; d(x,y) \leq \delta$

$\Rightarrow$ Approximate refinement: $x_p^A \overset{\boldsymbol{\delta}}{\approx} f(x_p^C)$

Property: $x_p^A \in S$
$\Rightarrow$ *Need to be **strengthened** to be preserved by approximation*



Approximation pattern

HS usually involve complex dynamics, hard to handle
$\Rightarrow$ *engineers use* ***approximation***

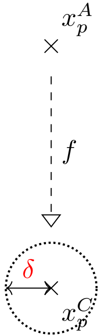
Approximation = substitute a system with a simpler system with a **similar behaviour** $\Rightarrow$ *formalise approximation as a* ***refinement***

Continuous refinement: $x_p^A = f(x_p^C)$

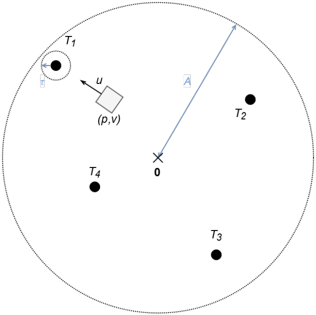
$\Rightarrow$ *Let's "loosen" equality:* $x \overset{\boldsymbol{\delta}}{\approx} y \;\equiv\; d(x,y) \leq \delta$

$\Rightarrow$ Approximate refinement: $x_p^A \overset{\boldsymbol{\delta}}{\approx} f(x_p^C)$

Property: $x_p^A \in S$
$\Rightarrow$ *Need to be* ***strengthened*** *to be preserved by approximation*
$\Rightarrow x_p^A \in S \land \forall \boldsymbol{\hat{x} \notin S, d(x_p^A, \hat{x}) > \delta}$ (shrinking $\mathcal{S}_\delta(S)$)



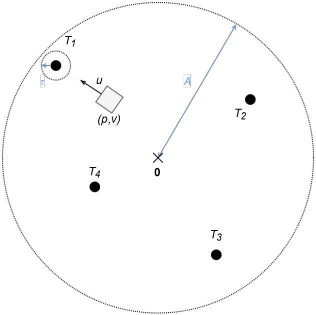
Approximation (Example)



- Robot $(p,\, v)$, visiting targets T_i
- Control system (u^C, w^C)
- Remains in area $\|p\| \leq A$
- Controller + motors $\Rightarrow$ *complex DE*

$$\begin{cases} \dot{v}^C & = & \frac{1}{2}u^C - K(p^C - w^C) - v^C \\ \dot{p}^C & = & v^C \\ \dot{w}^C & = & u^C \end{cases}$$

Approximation (Example)

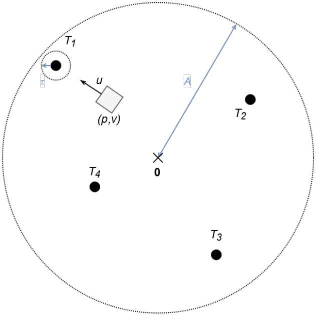


- Robot $(p,\ v)$, visiting targets T_i
- Control system (u^C, w^C)
- Remains in area $\|p\| \leq A$
- Controller + motors $\Rightarrow$ *complex DE*

$$\begin{cases} \dot{v}^C &= \frac{1}{2}u^C - K(p^C - w^C) - v^C \\ \dot{p}^C &= v^C \\ \dot{w}^C &= u^C \end{cases}$$

Idea: approximate system, simpler, $p^A = u^A$, with $p^A \overset{\delta}{\approx} p^C$
$+$ Safety : $\|p^A\| \leq A - \delta \ \Rightarrow\ \|p^C\| \leq A$

Approximation (Example)



- Robot $(p,\ v)$, visiting targets T_i
- Control system (u^C, w^C)
- Remains in area $\|p\| \leq A$
- Controller + motors $\Rightarrow$ *complex DE*

$$\begin{cases} \dot{v}^C &= \frac{1}{2}u^C - K(p^C - w^C) - v^C \\ \dot{p}^C &= v^C \\ \dot{w}^C &= u^C \end{cases}$$

Idea: approximate system, simpler, $p^A = u^A$, with $p^A \overset{\delta}{\approx} p^C$
$+$ Safety : $\|p^A\| \leq A - \delta \ \Rightarrow\ \|p^C\| \leq A$

Strategy: verified simpler model + correct approximation = preserved properties on complex model

Outline

1. Introduction
   - Context
   - Event-B and theories
2. Designing hybrid systems
   - Continuous behaviours in Event-B
   - Formal framework: principle and overview
   - Architectural patterns
   - Behavioural patterns
   - Co-verification, co-validation
3. Conclusion and Future Work

Co-validation – Motivation

Rodin (inc. Pro-B) = adapted to discrete systems, not so much for continuous...
$\Rightarrow$ *we should use adapted tools*

Co-validation – Motivation

Rodin (inc. Pro-B) = adapted to discrete systems, not so much for continuous...
$\Rightarrow$ *we should use adapted tools*

In particular, two specific POs:

$$\Gamma,\ \mathcal{I}([0,t] \lhd x_p),\ CBAP(t, t', x_p, x'_p, \mathcal{P}, \mathcal{H}) \vdash \mathcal{I}([t,t'] \lhd x'_p) \qquad \text{(CINV)}$$

$$\Gamma \vdash \exists t' \cdot t' \in \mathbb{R}^+ \wedge t' > t \wedge \mathbf{Feasible}([t,t'], x_p, \mathcal{P}, \mathcal{H}_{saf}) \qquad \text{(CFIS)}$$

Co-validation – Motivation

Rodin (inc. Pro-B) = adapted to discrete systems, not so much for continuous...
⇒ *we should use adapted tools*

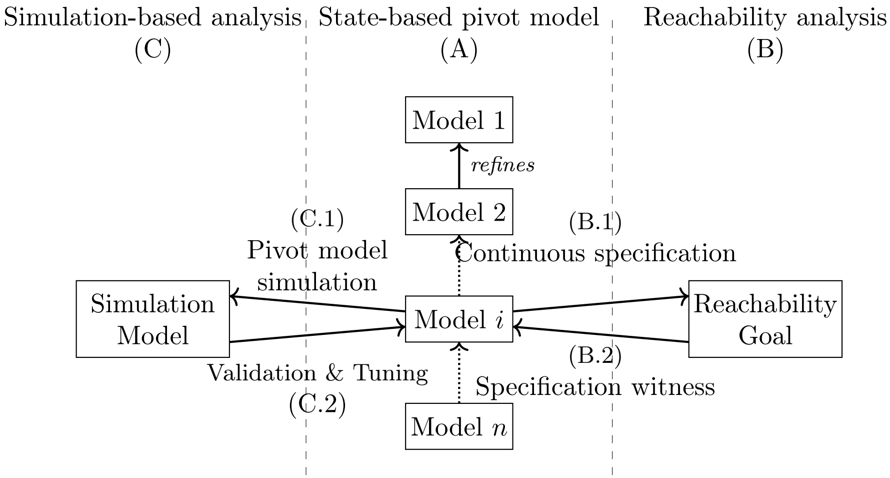
In particular, two specific POs:

$$\Gamma,\ \mathcal{I}([0,t] \lhd x_p),\ CBAP(t, t', x_p, x'_p, \mathcal{P}, \mathcal{H}) \vdash \mathcal{I}([t,t'] \lhd x'_p) \qquad \text{(CINV)}$$

$$\Gamma \vdash \exists t' \cdot t' \in \mathbb{R}^+ \wedge t' > t \wedge \mathbf{Feasible}([t,t'], x_p, \mathcal{P}, \mathcal{H}_{saf}) \qquad \text{(CFIS)}$$

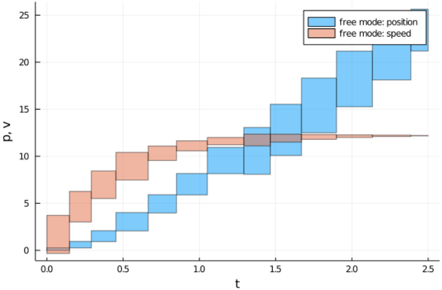
This correspond to a **reachability problem**

Principle


Simulation-based analysis
(C)
State-based pivot model
(A)
Reachability analysis
(B)
Model 1
refines
Model 2
(C.1)
Pivot model
simulation
(B.1)
Continuous specification
Simulation
Model
Model i
Reachability
Goal
(B.2)
Specification witness
Validation & Tuning
(C.2)
Model n

Some results

(case study = railway signaling systems)

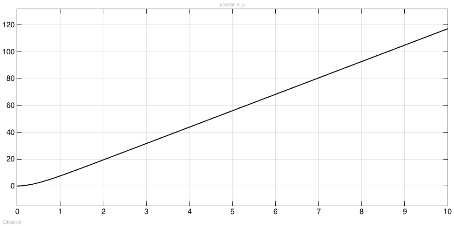


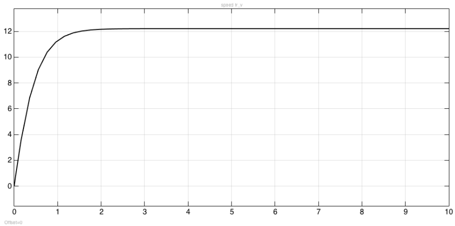
- using JuliaReach
- complex diff. eq.

$$\dot{v} = f - (a + bv + cv^2), \dot{p} = v$$

w/ invariant:
$p + StoppingDistance \leq EOA$

- simulation with Simulink





Outline

1 Introduction
- Context
- Event-B and theories

2 Designing hybrid systems
- Continuous behaviours in Event-B
- Formal framework: principle and overview
- Architectural patterns
- Behavioural patterns
- Co-verification, co-validation

3 Conclusion and Future Work

Conclusion

A formal framework for designing HS and CPS:

- **generic** and **reusable**
  $\Rightarrow$ *generic model + patterns defined once and for all, instantiation via refinement*
- integrates *discrete* and *continuous* aspects **at the same level**, integrates **domain knowledge**
  $\Rightarrow$ *thanks to the use of theories*
- features extensible architectural and behavioural **formal design patterns** $\Rightarrow$ *new pattern = refinement of the generic model*
- support of a general **methodology** for HS development
  $\Rightarrow$ *concrete system = sequence of pattern application with generic model as root*

Note: a diversity of case studies available on my website *https://irit.fr/~Guillaume.Dupont/models/*

Future work

Include more types of systems:

- other architectures, other dynamics
- other domains + properties

Possible improvements

- easing modelling *(models a bit difficult to write...)*
- helping proof *(proof automation, specialised provers)*

Bridging the gap with implementation:

- **discretisation**, floating points
- event-based > clock-based, heterogeneous times
- constraint synthesis

Part II

Bibliography

References I

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. 1st. New York, NY, USA: Cambridge University Press, 2010. ISBN: 9781139637794.

[Alu+95] Rajeev Alur et al. "The algorithmic analysis of hybrid systems". In: *Theoretical Computer Science* 138.1 (1995). Hybrid Systems, pp. 3–34. ISSN: 0304-3975.

[BM13] Michael Butler and Issam Maamria. "Practical Theory Extension in Event-B". In: *Theories of Programming and Formal Methods*. Ed. by Zhiming Liu, Jim Woodcock, and Huibiao Zhu. Vol. 8051. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 67–81. ISBN: 978-3-642-39697-7.