Formal Specification and Verification

Deductive Verification: An introduction (2)

29.07.2014

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

Extensions in two possible directions:

- More precise description of the actions/events
  - Propositional Dynamic Logic (last time)
  - Hoare logic (not discussed in this lecture)
- More precise description of states (and possibly also of actions)
  - succinct representation: formulae represent a set of states
  - deductive verification (today)

Last time

Transition systems revisited

Program graphs

From program graphs to transition systems

Set of states: $S = Loc \times Eval(Var)$

Problem

$Eval(Var)$ can be very large

(some variables can have values in large data domains e.g. integers)

Therefore it is difficult to concretely represent $\rightarrow$

(the relation usually very large as well)

Solution

Succinct representation of sets of states and of transitions between states

- Set of states: Formula (property of all states in the set)
- Transitions: Formulae (relation between the old values of the variables and the new values of the variables)

Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

States:

(l, β), where l location and β assignment of values to the variables.

Idea: Take into account an additional variable pc (program counter), having as domain the set of locations.

State: assignment of values to the variables and to pc

Set of states: Logical formula

Example:

$y \geq z$: The set of all states (l, β) for which $\beta(y) \geq \beta(z)$ (i.e. $\beta \models y \geq z$)

Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
       x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

Transition relation: $(l, \beta) \to (l', \beta')$

Expressed by logical formulae: Formula containing primed and unprimed variables.

Example:

- $\rho_1 = (move(l_1, l_2) \wedge y \geq z \wedge skip(x, y, z))$
- $\rho_2 = (move(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$
- $\rho_3 = (move(l_2, l_3) \wedge x \geq y \wedge skip(x, y, z))$
- $\rho_4 = (move(l_3, l_4) \wedge x \geq z \wedge skip(x, y, z))$
- $\rho_5 = (move(l_3; l_5) \wedge x + 1 \leq z \wedge skip(x, y, z))$

Abbreviations:

$move(l, l') := (pc = l \wedge pc' = l')$

$skip(v_1, \dots, v_n) := (v_1' = v_1 \wedge \dots \wedge v_n' = v_n)$

Programs as transition systems

Verification problem: Program + Description of the "bad" states

Succinct representation:

$$P = (Var, pc, Init, \mathcal{R}) \qquad \phi_{\mathrm{err}}$$

- V - finite (ordered) set of program variables
- pc - program counter variable (pc included in V)
- $Init$ - initiation condition given by formula over V
- $\mathcal{R}$ - a finite set of transition relations

  Every transition relation $\rho \in \mathcal{R}$ is given by a formula over the variables V and their primed versions V'

- ϕ_{err} - an error condition given by a formula over V

States, sets and relations

- Each program variable x is assigned a domain of values D_x.
- Program state = function that assigns each program variable a value from its respective domain
- S = set of program states
- Formula with free variables in V = set of program states
- Formula with free variables in V and V' = binary relation over program states
  - First component of each pair refers to values of the variables V
  - Second component of the pair refers to values of the variables V' (typically the new variables of the variables in V after an instruction was executed)

States, sets and relations

- We identify formulas with the sets and relations that they represent
- We identify the entailment relation between formulas $\models$ with set inclusion
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

States, sets and relations

- We identify formulas with the sets and relations that they represent
- We identify the entailment relation between formulas $\models$ with set inclusion
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

Example:

- Formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z
- Formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1

States, sets and relations

- We identify formulas with the sets and relations that they represent
- We identify the entailment relation between formulas $\models$ with set inclusion
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

Example:

- Formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z
- Formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1
- If program state s assigns 1, 3, 2, and l_1 to program variables x, y, z, and pc, respectively, then $s \models y \geq z$

States, sets and relations

- We identify formulas with the sets and relations that they represent
- We identify the entailment relation between formulas $\models$ with set inclusion
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

Example:

- Formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z
- Formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1
- If program state s assigns 1, 3, 2, and l_1 to program variables x, y, z, and pc, respectively, then $s \models y \geq z$
- Logical consequence: $y \geq z \models y + 1 \models z$

Example Program

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
       x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

Example program

- Program variables $V = (pc, x, y, z)$
- Program counter pc
- Program variables x, y, and z range over integers: $D_x = D_y = D_z = \text{Int}$
  Program counter pc ranges over control locations: $D_{pc} = L$
- Set of control locations $L = \{l_1, l_2, l_3, l_4, l_5\}$
- Initiation condition $\mathit{Init} := (pc = l_1)$
- Error condition $\phi_{\text{err}} := (pc = l_5)$
- Program transitions $\mathcal{R} = \{\rho_1, \ldots, \rho_5\}$, where:

$$\rho_1 = (\mathit{move}(l_1, l_2) \wedge y \geq z \wedge \mathit{skip}(x, y, z))$$
$$\rho_2 = (\mathit{move}(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge \mathit{skip}(y, z))$$
$$\rho_3 = (\mathit{move}(l_2, l_3) \wedge x \geq y \wedge \mathit{skip}(x, y, z))$$
$$\rho_4 = (\mathit{move}(l_3, l_4) \wedge x \geq z \wedge \mathit{skip}(x, y, z))$$
$$\rho_5 = (\mathit{move}(l_3; l_5) \wedge x + 1 \leq z \wedge \mathit{skip}(x, y, z))$$

Initial state, error state, transition relation

- Each state that satisfies the initiation condition *Init* is called an initial state
- Each state that satisfies the error condition *err* is called an error state
- Program transition relation $\rho_{\mathcal{R}}$ is the union of the single-statement transition relations (formula representation: disjunction) i.e.,

$$\rho_{\mathcal{R}} = \bigvee_{\rho \in \mathcal{R}} \rho$$

- The state s has a transition to the state s' if the pair of states (s, s') lies in the program transition relation $\rho_{\mathcal{R}}$, i.e., if $(s, s') \models \rho_{\mathcal{R}}$:
  - $s : V \rightarrow \bigcup_{x \in V} D_x, \quad s(x) \in D_x$ for all $x \in V$
  - $s' : V' \rightarrow \bigcup_{x \in V} D_x, \quad s(x') \in D_x$ for all $x \in V$
  - $\beta : V \cup V' \bigcup_{x \in X} D_x$ defined for every $x \in V$ by $\beta(x) = s(x), \beta(x') = s'(x)$ has the property that $\beta \models \rho_{\mathcal{R}}$

Computation

A program computation is a sequence of states $s_1 s_2 \ldots$ such that:

- The first element is an initial state, i.e., $s_1 \models \mathit{Init}$
- Each pair of consecutive states (s_i, s_{i+1}) is connected by a program transition, i.e., $(s_i, s_{i+1}) \models \rho_{\mathcal{R}}$.
- If the sequence is finite then the last element does not have any successors i.e., if the last element is s_n, then there is no state s such that $(s_n, s) \models \rho_{\mathcal{R}}$.

Example Program

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

Example of a computation:

$$(l_1, 1, 3, 2), (l_2, 1, 3, 2), (l_2, 2, 3, 2), (l_2, 3, 3, 2), (l_3, 3, 3, 2), (l_4, 3, 3, 2)$$

- sequence of transitions $\rho_1, \rho_2, \rho_2, \rho_3, \rho_4$
- state $=$ tuple of values of program variables pc, x, y, and z
- last program state does not any successors

Correctness: Safety

- a state is reachable if it occurs in some program computation
- a program is safe if no error state is reachable
- ... if and only if no error state lies in ϕ_{reach},

$$\phi_{\text{err}} \land \phi_{\text{reach}} \models \perp$$

where ϕ_{reach} = set of program states which are reachable from some initial state

- ... if and only if no initial state lies in $\phi_{reach^{-1}}$,

$$\mathit{Init} \land \phi_{\text{reach}^{-1}}(\phi_{\text{err}}) \models \perp$$

where $\phi_{\text{reach}^{-1}}(\phi_{\text{err}})$ = set of program states from which some state in ϕ_{err} is reachable

Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

Set of reachable states:

$$
\begin{aligned}
\phi_{reach} \quad = \quad & (pc = l_1 \vee \\
& (pc = l_2 \wedge y \geq z) \vee \\
& (pc = l_3 \wedge y \geq z \wedge x \geq y) \vee \\
& (pc = l_4 \wedge y \geq z \wedge x \geq y)
\end{aligned}
$$

Post operator

Let ϕ be a formula over V

Let ρ be a formula over V and V'

Define a post-condition function *post* by:

$$post(\phi, \rho) = \exists V'' : \phi[V''/V] \wedge \rho[V''/V][V/V']$$

An application $post(\phi, \rho)$ computes the image of the set ϕ under the relation ρ.

Post operator

Let ϕ be a formula over V

Let ρ be a formula over V and V'

Define a post-condition function *post* by:

$$post(\phi, \rho) = \exists V'' : \phi[V''/V] \wedge \rho[V''/V][V/V']$$

An application $post(\phi, \rho)$ computes the image of the set ϕ under the relation ρ.

post distributes over disjunction wrt. each argument:

- $post(\phi, \rho_1 \vee \rho_2) = post(\phi, \rho_1) \vee post(\phi, \rho_2)$
- $post(\phi_1 \vee \phi_2, \rho) = post(\phi_1, \rho) \vee post(\phi_2, \rho)$

Application of post in example program

Set of states $\phi := (pc = l_2 \wedge y \geq z)$

Transition relation $\rho := \rho_2$

$$\rho_2 = (move(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$$

$$\begin{aligned}
post(\phi, \rho) &= \exists V''(pc = l_2 \wedge y \geq x)[V''/V] \wedge \rho_2[V''/V][V/V'] \\
&= \exists V''(pc'' = l_2 \wedge y'' \geq x'') \wedge \\
&\quad (pc'' = l_2 \wedge pc' = l_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge y' = y'' \wedge z' = z'')[V/ \\
&= \exists V''(pc'' = l_2 \wedge y'' \geq x'') \wedge \\
&\quad (pc'' = l_2 \wedge pc = l_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge y = y'' \wedge z = z'') \\
&= (pc = l_2 \wedge y \leq z \wedge x \leq y)
\end{aligned}$$

Application of post in example program

Set of states $\phi := (pc = l_2 \wedge y \geq z)$

Transition relation $\rho := \rho_2$

$$\rho_2 = (move(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$$

$$\begin{aligned}
post(\phi, \rho) &= \exists V''(pc = l_2 \wedge y \geq x)[V''/V] \wedge \rho_2[V''/V][V/V'] \\
&= \exists V''(pc'' = l_2 \wedge y'' \geq x'') \wedge \\
&\quad (pc'' = l_2 \wedge pc' = l_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge y' = y'' \wedge z' = z'')[V/ \\
&= \exists V''(pc'' = l_2 \wedge y'' \geq x'') \wedge \\
&\quad (pc'' = l_2 \wedge pc = l_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge y = y'' \wedge z = z'') \\
&= (pc = l_2 \wedge y \leq z \wedge x \leq y)
\end{aligned}$$

[Renamed] program variables:
$V = (pc, x, y, z)$, $V' = (pc', x', y', z')$, $V'' = (pc'', x'', y'', z'')$

Iteration of post

$post^n(\phi, \rho) =$ n-fold application of post to ϕ under ρ

$$post^n(\phi, \rho) = \begin{cases} \phi & \text{if } n = 0 \\ post(post^{n-1}(\phi, \rho)), \rho) & \text{otherwise} \end{cases}$$

Characterize ϕ_{reach} using iterates of post:

$$\begin{aligned} \phi_{\text{reach}} &= \text{Init} \vee post(Init, \rho_{\mathcal{R}}) \vee post(post(Init, \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \vee \ldots \\ &= \bigvee_{i \geq 0} post^i(Init, \rho_{\mathcal{R}}) \end{aligned}$$

disjuncts = iterates for every natural number n ("ω-iteration")

Finite iteration post may suffice

Fixpoint reached in n steps if $\bigvee_{i=1}^{n} post^i(Init, \rho_{\mathcal{R}}) = \bigvee_{i=1}^{n+1} post^i(Init, \rho_{\mathcal{R}})$

Then $\bigvee_{i=1}^{n} post^i(Init, \rho_{\mathcal{R}}) = \bigvee_{i \geq 0} post^i(Init, \rho_{\mathcal{R}})$

Forward reachability analysis

Compute $\bigvee_{i=1}^{n} post^i(Init, \rho_{\mathcal{R}})$, $n \geq 0$.

If there exists $m \in \mathbb{N}$ such that

$$\bigvee_{i=1}^{n} post^i(Init, \rho_{\mathcal{R}}) = \bigvee_{i=1}^{n+1} post^i(Init, \rho_{\mathcal{R}})$$

then fixpoint reached.

Let $\phi_{\text{reach}} := \bigvee_{i=1}^{n} post^i(Init, \rho_{\mathcal{R}})$

If $\phi_{\text{reach}} \cap \phi_{\text{err}} = \emptyset$ then safety is guaranteed.

Backward reachability analysis

Another possibility: Start from a bad state and compute states from which the bad state can be reached.

If the initial states are not among these states then safety is guaranteed.

Pre operator

Let ϕ be a formula over V

Let ρ be a formula over V and V'

Define a pre-condition function *pre* by:

$$pre(\phi, \rho) = \exists V' : \rho \wedge \phi[V'/V]$$

An application $pre(\phi, \rho)$ computes the preimage of the set ϕ under the relation ρ.

Computation of pre^n similar.

Problem

Reasoning modulo theories

Reasoning modulo theories

Goal: Devise efficient methods for reasoning modulo theories

SAT checking (can reduce entailment to checking satisfiability)

Example:

Check whether conjunctions of constraints in linear arithmetic is satisfiable: classical methods exist, e.g. simplex.

Check whether a conjunction of equalities and disequalities of ground terms is satisfiable: methods exist (e.g. congruence closure)

Challenge: efficient methods for handling arbitrary Boolean combinations of constraints in such theories.

Possible solution: Extend the DPLL method to reasoning modulo theories.

Reminder: The DPLL algorithm

State: $M||F$,

where:

- M partial assignment (sequence of literals),

  some literals are annotated (L^d: decision literal)

- F clause set.

A succinct formulation

UnitPropagation

$M||F, C \vee L \Rightarrow M, L||F, C \vee L$ if $M \models \neg C$, and L undef. in M

Decide

$M||F \Rightarrow M, L^d||F$ if L or $\neg L$ occurs in F, L undef. in M

Fail

$M||F, C \Rightarrow$ Fail if $M \models \neg C$, M contains no decision literals

Backjump

$$M, L^d, N||F \Rightarrow M, L'||F \quad \text{if} \begin{cases} \text{there is some clause } C \vee L' \text{ s.t.:} \\ F \models C \vee L', M \models \neg C, \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{cases}$$

SAT Modulo Theories (SMT)

Some problems are more naturally expressed in richer logics than just propositional logic, e.g:

- Software/Hardware verification needs reasoning about equality, arithmetic, data structures, ...

SMT consists of deciding the satisfiability of a ground 1st-order formula with respect to a background theory T

SAT Modulo Theories (SMT)

The "very eager" approach to SMT

Method:

– translate problem into equisatisfiable propositional formula;

– use off-the-shelf SAT solver

- Why "eager"?

  Search uses all theory information from the beginning

- Characteristics:

  + Can use best available SAT solver

  – Sophisticated encodings are needed for each theory

  – Sometimes translation and/or solving too slow

Main Challenge for alternative approaches is to combine:

- DPLL-based techniques for handling the boolean structure

- Efficient theory solvers for conjunctions of $\mathcal{T}$-literals

SAT Modulo Theories (SMT)

"Lazy" approaches to SMT: Idea

Example: consider $\mathcal{T} = UIF$ and the following set of clauses:

$$\underbrace{f(g(a)) \not\approx f(c)}_{\neg P_1} \vee \underbrace{g(a) \approx d}_{P_2}, \quad \underbrace{g(a) \approx c}_{P_3}, \quad \underbrace{c \not\approx d}_{\neg P_4}$$

1. Send $\{\neg P_1 \vee P_2,\ P_3,\ \neg P_4\}$ to SAT solver

   SAT solver returns model $[\neg P_1, P_3, \neg P_4]$
   Theory solver says $\neg P_1 \wedge P_3 \wedge \neg P_4$ is $\mathcal{T}$-inconsistent

2. Send $\{\neg P_1 \vee P_2,\ P_3,\ \neg P_4,\ P_1 \vee \neg P_3 \vee P_4\}$ to SAT solver

   SAT solver returns model $[P_1, P_2, P_3, \neg P_4]$
   Theory solver says $P_1 \wedge P_2 \wedge P_3 \wedge \neg P_4$ is $\mathcal{T}$-inconsistent

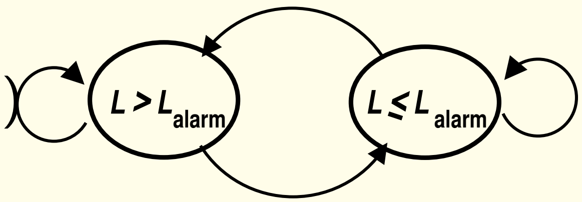
3. Send $\{\neg P_1 \vee P_2, P_3, \neg P_4, P_1 \vee \neg P_3 \vee P_4, \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4\}$ to SAT solver

   SAT solver says UNSAT

Example

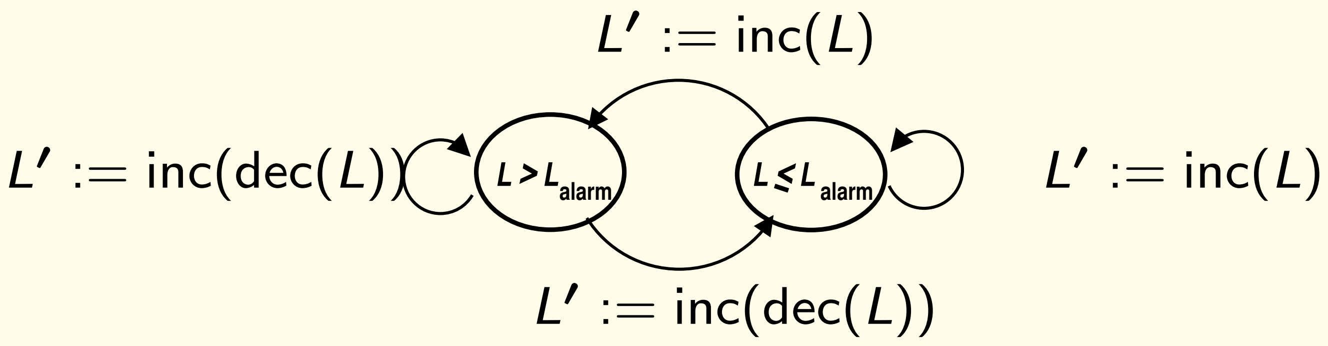
Example: Water tank

$L' := \text{inc}(L)$

$L' := \text{inc}(\text{dec}(L))$  $L' := \text{inc}(L)$

$L' := \text{inc}(\text{dec}(L))$

Example: Water tank



The system has two possible transitions which can be represented as

$$(T_1) \quad L > L_{alarm} \rightarrow L := \mathrm{inc}(\mathrm{dec}(L))$$
$$(T_2) \quad L \leq L_{alarm} \rightarrow L := \mathrm{inc}(L),$$

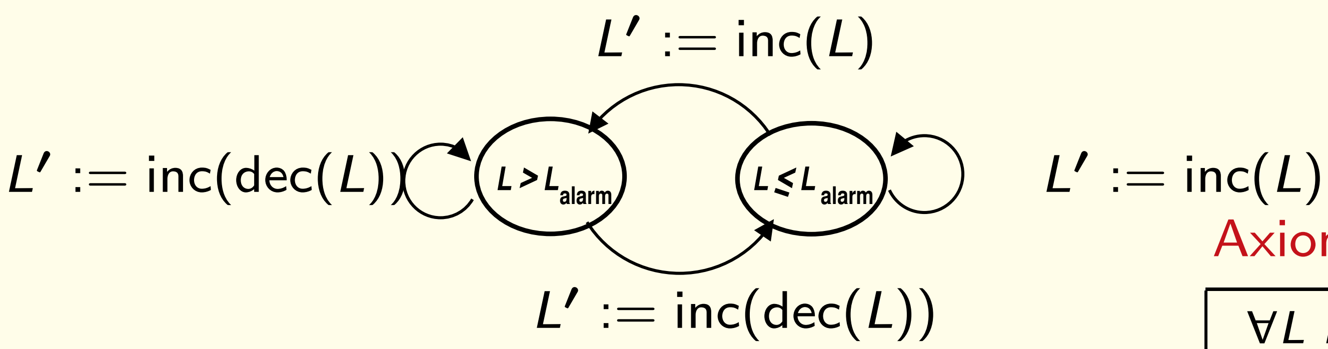
We assume that the watertank starts with an initial condition

$$(\mathrm{init}) \quad L \leq L_{alarm},$$

and safety of the system is given by

$$(\mathrm{safe}) \quad L \leq L_{overflow}.$$

Example: Water tank



The system has two possible transitions which can be represented as

$$(T_1) \quad L > L_{alarm} \rightarrow L := \mathrm{inc}(\mathrm{dec}(L))$$
$$(T_2) \quad L \leq L_{alarm} \rightarrow L := \mathrm{inc}(L),$$

We assume that the watertank starts with an initial condition

$$(\mathrm{init}) \quad L \leq L_{alarm},$$

and safety of the system is given by

$$(\mathrm{safe}) \quad L \leq L_{overflow}.$$

Axioms: Variant 1

$$\forall L \; L \leq \mathrm{inc}(L)$$
$$\forall L \; \mathrm{dec}(L) \leq L$$
$$\forall L \; \mathrm{inc}(\mathrm{dec}(L)) \leq L$$
$$\forall L \; L \leq L_{alarm} \rightarrow \mathrm{inc}(L) \leq L_{overflow}$$
$$L_{alarm} \leq L_{overflow}$$

Axioms: Variant 2

$$\forall L \quad L \leq \mathrm{inc}(L)$$
$$\forall L \quad \mathrm{dec}(L) \leq L$$
$$\forall L \quad \mathrm{inc}(\mathrm{dec}(L)) \leq L$$
$$\forall L, L' \quad L \leq L' \rightarrow \mathrm{inc}(L) \leq \mathrm{inc}(L'$$
$$\mathrm{inc}(L_{alarm}) \leq L_{overflow}$$

Example: Water tank

Model checking starts with the representation of unsafe states

$$\neg\mathsf{safe} \equiv L > L_{overflow}$$

The pre-states of ¬safe are given by:

$$\mathsf{Pre}(\neg\mathsf{safe}) \equiv \neg\mathsf{safe} \vee (G_1 \wedge (\neg\mathsf{safe})\sigma_1) \vee (G_2 \wedge (\neg\mathsf{safe})\sigma_2),$$

or spelled out

$$\begin{aligned} \mathsf{Pre}(\neg\mathsf{safe}) \equiv\ & L > L_{overflow} \\ & \vee (L > L_{alarm} \wedge \mathsf{inc}(\mathsf{dec}(L)) > L_{overflow}) \\ & \vee (L \leq L_{alarm} \wedge \mathsf{inc}(L) > L_{overflow}). \end{aligned}$$

At this point, we should be able to verify that $\mathsf{Pre}(\neg\mathsf{safe}) \wedge \mathsf{init} \models_{\mathcal{T}} \Box$, and that $\models_{\mathcal{T}} \mathsf{Pre}(\neg\mathsf{safe}) \rightarrow \neg\mathsf{safe}$. Together, this implies safety of the simple watertank.

We solved both these proof tasks using H-PiLoT. (0.080987s/0.12798s)

Example: Water tank

Discussing the example

- Type of properties not included in the type of problems the present implementation of FOMC can handle

$$\begin{array}{rcl} & & \mathrm{inc}(\mathrm{dec}(L)) \leq L \\ L > L_{alarm} & \rightarrow & \mathrm{inc}(L) \leq L_{overflow} \end{array}$$

- Theories needed in verification problems could be proved to be well-behaved (with or without monotonicity of $\mathrm{inc}, \mathrm{dec}$).

Example: Water tank

Also studied: variant of previous example: watertank with delay

- Reaction not immediate ("old" update rule maintained once more)

- Changed description of transitions

To ensure safety we need slightly different properties, including

(5′) $\forall L \;\; L \leq L_{alarm} \rightarrow \text{inc}(\text{inc}(L)) \leq L_{overflow}$

FOMC: Two iterations needed (Can also be proved by HPiLoT)

Note: to simplify proof tasks use $\text{Pre}^2(\neg\text{safe}) = \bigvee \phi_i$;
check $\phi_i \wedge \neg\text{Pre}(\neg\text{safe}) \models \perp$ for all i

Problems

It is not guaranteed that the fixpoint is reached in a finite/bounded number of steps.

Problems

It is not guaranteed that the fixpoint is reached in a finite/bounded number of steps.

Need to analyze alternative solutions

Verification

Modeling/Formalization

System Specification

Is the system safe?

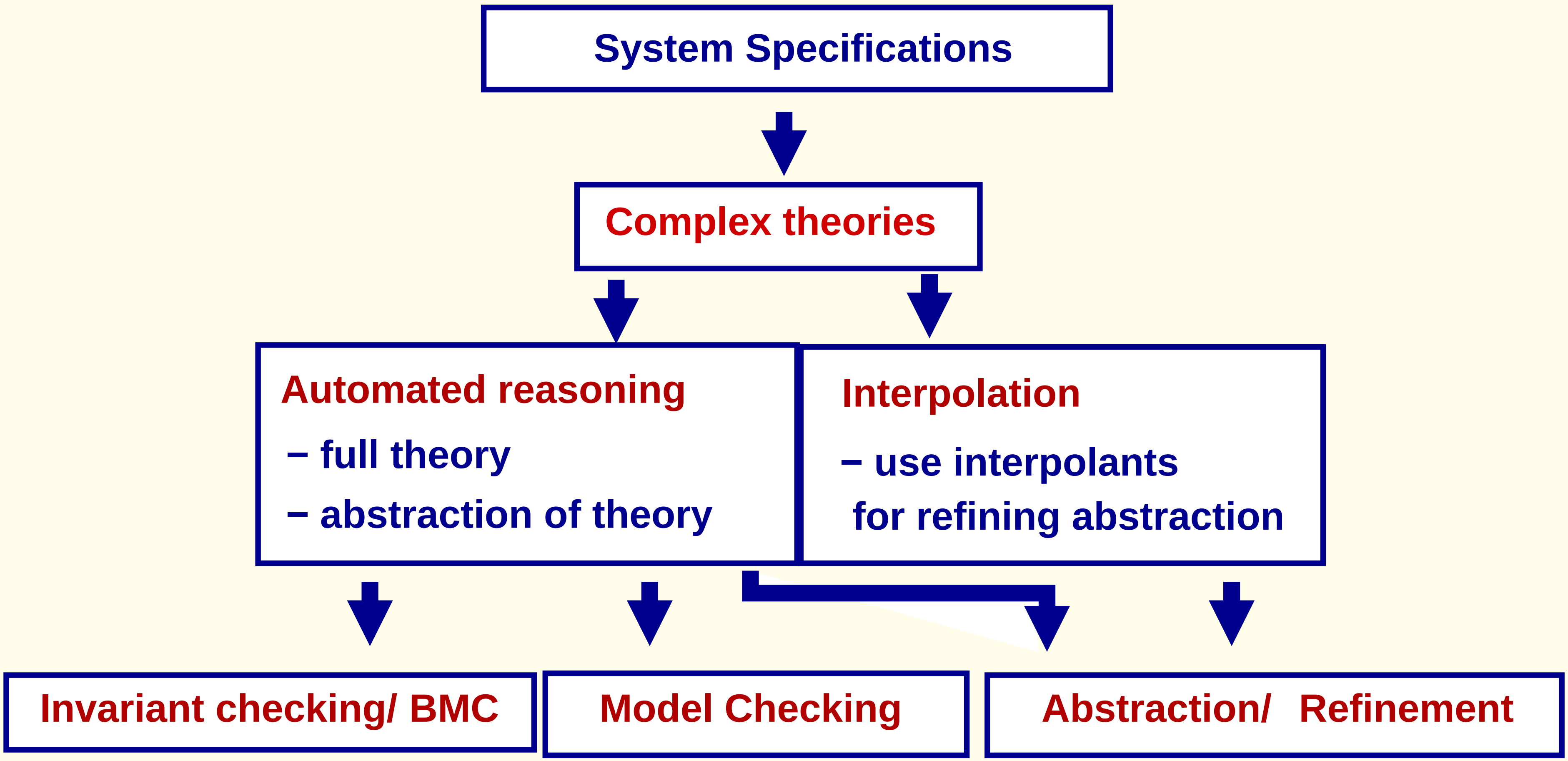
Is safety guaranteed on all paths
of length < n which start in an initial state?

Is the safety property an invariant of the system?
Can we generate an invariant which implies safety?

Invariant checking/ BMC	Model Checking	Abstraction/ Refinement

Verification

Modeling/Formalization



Invariant checking; Bounded model checking

S specification $\mapsto \Sigma_S$ signature of S; $\mathcal{T}_S$ theory of S; T_S transition system $\mathrm{Init}(\overline{x})$; $\rho_{\mathcal{R}}(\overline{x}, \overline{x}')$

Given: $\mathrm{Safe}(x)$ formula (e.g. safety property)

- **Invariant checking**

  (1) $\mathcal{T}_S \models \mathrm{Init}(\overline{x}) \rightarrow \mathrm{Safe}(\overline{x})$ (Safe holds in the initial state)

  (2) $\mathcal{T}_S \models \mathrm{Safe}(\overline{x}) \wedge \rho_{\mathcal{R}}(\overline{x}, \overline{x}') \rightarrow \mathrm{Safe}(\overline{x}')$ (Safe holds before $\Rightarrow$ holds after update)

- **Bounded model checking (BMC):**

  Check whether, for a fixed k, unsafe states are reachable in at most k steps, i.e. for all $0 \leq j \leq k$:

$$\mathcal{T}_S \models \mathrm{Init}(x_0) \wedge \rho_{\mathcal{R}}(x_0, x_1) \wedge \cdots \wedge \rho_{\mathcal{R}}(x_{j-1}, x_j) \wedge \neg \mathrm{Safe}(x_j) \rightarrow \perp$$

Reasoning modulo theories

Goal: Devise efficient methods for reasoning modulo theories

Problems

− First order logic is undecidable

− In applications, theories do not occur alone

$\mapsto$ need to consider combinations of theories

+ Fragments of theories occurring in applications are often decidable

+ Often provers for the component theories can be combined efficiently

Probleme

- First order logic is undecidable
- In applications, theories do not occur alone

  $\mapsto$ need to consider combinations of theories

+ Fragments of theories occurring in applications are often decidable

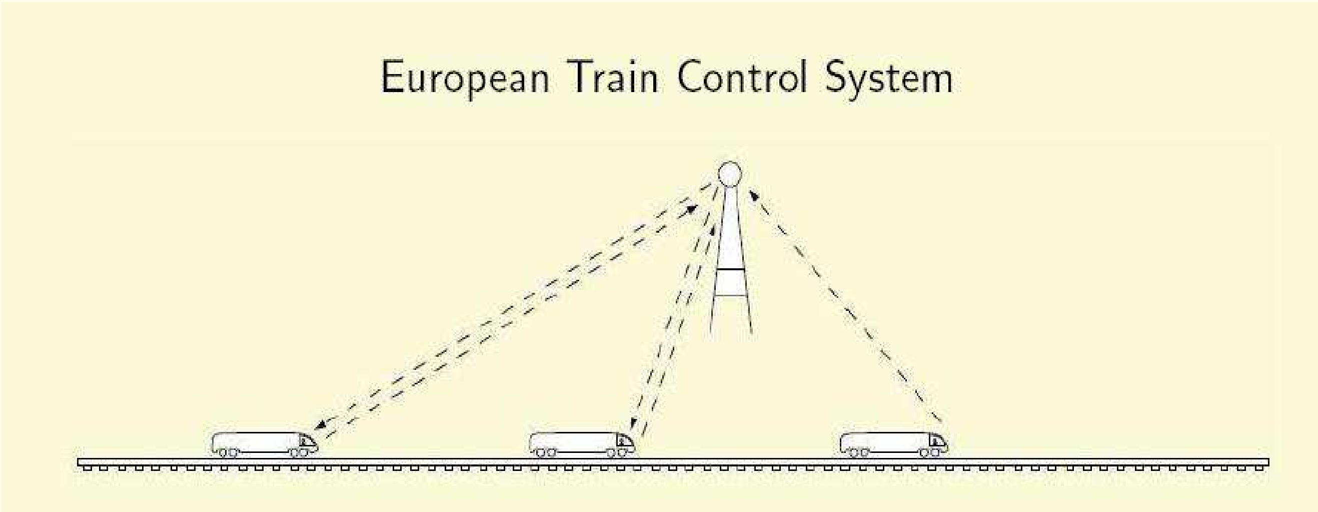
+ Often provers for the component theories can be combined efficiently

Important goals:

- Identify decidable theories which are important in applications (Extensions/Combinations) possibly with low complexity
- Development & Implementation of efficient Decision Procedures

Example: ETCS Case Study (AVACS project)

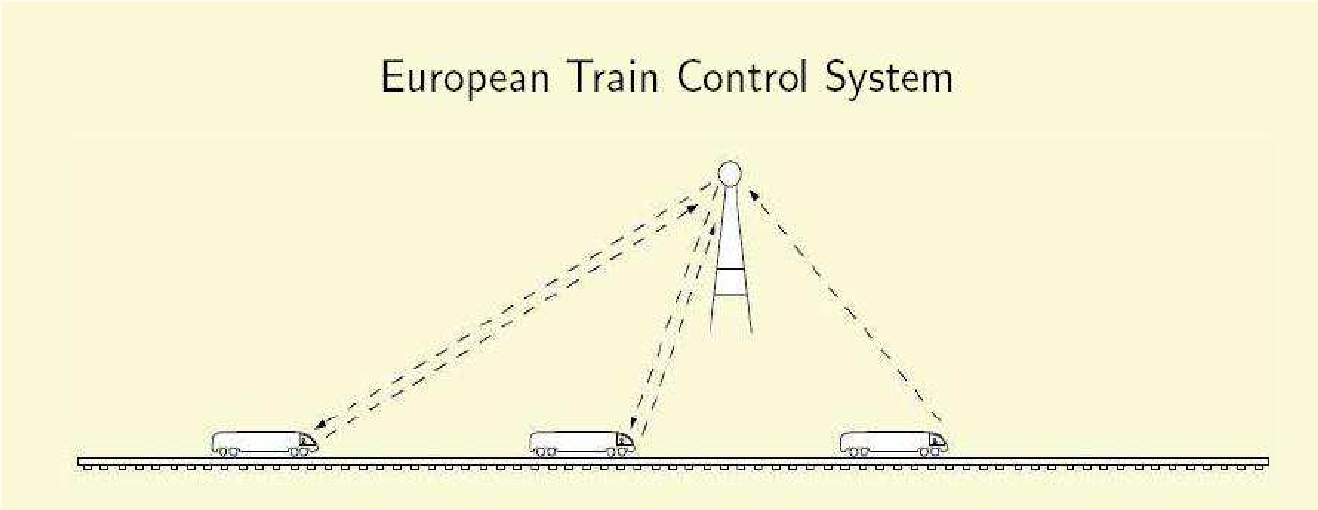
Simplified version of ETCS Case Study [Jacobs,VS'06, Faber,Jacobs,VS'07]



Number of trains:	$n \geq 0$	$\mathbb{Z}$
Minimum and maximum speed of trains:	$0 \leq \mathsf{min} < \mathsf{max}$	$\mathbb{R}$
Minimum secure distance:	$l_{\mathsf{alarm}} > 0$	$\mathbb{R}$
Time between updates:	$\Delta t > 0$	$\mathbb{R}$
Train positions before and after update:	$pos(i), pos'(i)$	$: \mathbb{Z} \rightarrow \mathbb{R}$

Example: ETCS Case Study (AVACS project)

Simplified version of ETCS Case Study [Jacobs,VS'06, Faber,Jacobs,VS'07]



Update(pos, pos′) :

- $\forall i \; (i = 0 \rightarrow pos(i) + \Delta t {*} \text{min} \leq pos'(i) \leq pos(i) + \Delta t {*} \text{max})$
- $\forall i \; (0 < i < n \;\wedge\; pos(i-1) > 0 \;\wedge\; pos(i-1) - pos(i) \geq l_{\text{alarm}}$
  $\rightarrow pos(i) + \Delta t * \text{min} \leq pos'(i) \leq pos(i) + \Delta t {*} \text{max})$

...

Example: ETCS Case Study (AVACS project)

Safety property: No collisions $\quad \mathsf{Safe}(\mathsf{pos}) : \quad \forall i, j (i < j \rightarrow \mathsf{pos}(i) > \mathsf{pos}(j))$

$$\boxed{\textbf{Inductive invariant:} \quad \mathsf{Safe}(\mathsf{pos}) \wedge \mathsf{Update}(\mathsf{pos}, \mathsf{pos}') \wedge \neg \mathsf{Safe}(\mathsf{pos}') \models_{\mathcal{T}_S} \perp}$$

where $\mathcal{T}_S$ is the extension of the (disjoint) combination $\mathbb{R} \cup \mathbb{Z}$
with two functions, $\mathsf{pos}, \mathsf{pos}' : \mathbb{Z} \rightarrow \mathbb{R}$

Problem: Satisfiability test for quantified formulae in complex theory

More complex ETCS Case studies

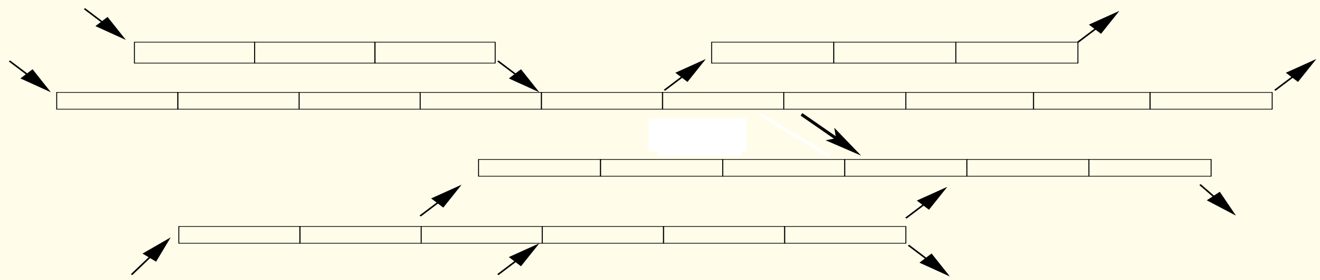
[Faber, Jacobs, VS, 2007]

- Take into account also:
  - Emergency messages
  - Durations
- Specification language: CSP-OZ-DC
  - Reduction to satisfiability in theories for which decision procedures exist
- **Tool chain:** [Faber, Ihlemann, Jacobs, VS]

CSP-OZ-DC $\mapsto$ Transition constr. $\mapsto$ Decision procedures (H-PILoT)

Example 2: Parametric topology

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]

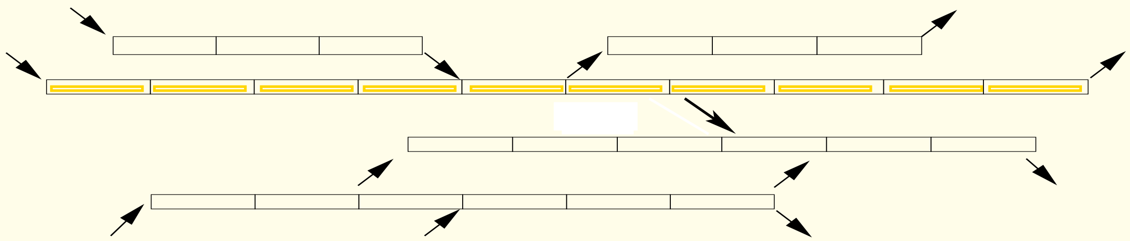


Assumptions:

- No cycles
- in-degree (out-degree) of associated graph at most 2.

Parametricity and modularity

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]



Assumptions:

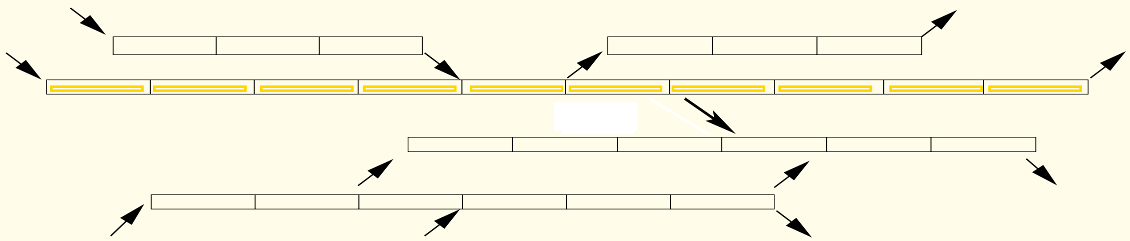
- No cycles
- in-degree (out-degree) of associated graph at most 2.

Approach:

- Decompose the system in trajectories (linear rail tracks; may overlap)
- Task 1: - Prove safety for trajectories with incoming/outgoing trains
  - Conclude that for control rules in which trains have sufficient freedom (and if trains are assigned unique priorities) safety of all trajectories implies safety of the whole system
- Task 2: - General constraints on parameters which guarantee safety

Parametricity and modularity

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]

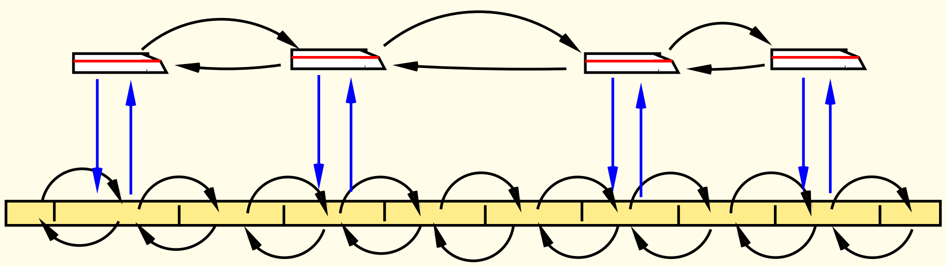


Assumptions:

- No cycles
- in-degree (out-degree) of associated graph at most 2.

Data structures:

- 2-sorted pointers
  - p_1: trains
  - p_2: segments



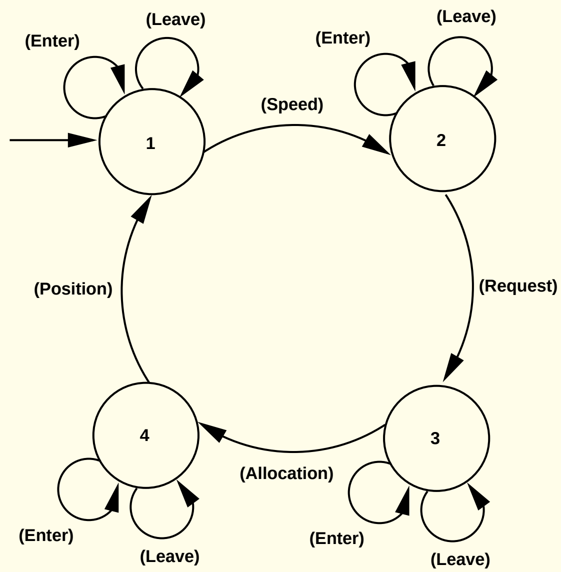
- scalar fields (f:$p_i \rightarrow \mathbb{R}$, g:$p_i \rightarrow \mathbb{Z}$)
- updates efficient decision procedures (H-PiLoT)

Example: Controller for line track (RBC)

CSP part: specifies the processes and their interdependency.

The RBC system passes repeatedly through four phases, modeled by events:

- updSpd (speed update)
- req (request update)
- alloc (allocation update)
- updPos (position update)



Between these events, trains may leave or enter the track (at specific segments), modeled by the events leave and enter.

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the Init formulae, update rules.

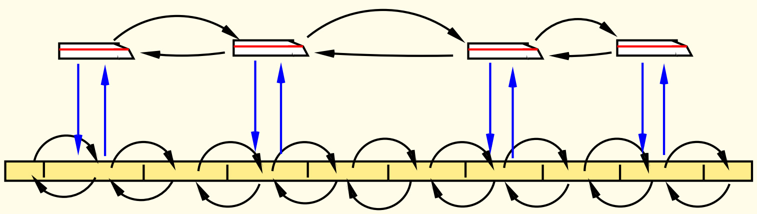
- 1. Data classes declare function symbols that can change their values during runs of the system

  Data structures:

  - 2-sorted pointers

    train: trains

    segm: segments



Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the Init formulae, update rules.

- 1. Data classes declare function symbols that can change their values during runs of the system, and are used in the OZ part of the specification.

- 2. Axioms: define properties of the data structures and system parameters which do not change
  - $\mathit{gmax} : \mathbb{R}$ (the global maximum speed),
  - $\mathit{decmax} : \mathbb{R}$ (the maximum deceleration of trains),
  - $d : \mathbb{R}$ (a safety distance between trains),
  - Properties of the data structures used to model trains/segments

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the Init formulae, update rules.

- 3. Init schema. describes the initial state of the system.
  - trains - doubly-linked list; placed correctly on the track segments
  - all trains respect their speed limits.

- 4. Update rules specify updates of the state space executed when the corresponding event from the CSP part is performed.
  Example: Speed update

Modular Verification

COD specification	$\mapsto \Sigma_S$ signature of S; $\mathcal{T}_S$ theory of S; T_S transition constraint system $\mathsf{Init}(\overline{x})$; $\mathsf{Update}(\overline{x}, \overline{x}')$

Given: $\mathsf{Safe}(x)$ formula (e.g. safety property)

- **Invariant checking**

  (1) $\models_{\mathcal{T}_S} \mathsf{Init}(\overline{x}) \rightarrow \mathsf{Safe}(\overline{x})$ (Safe holds in the initial state)

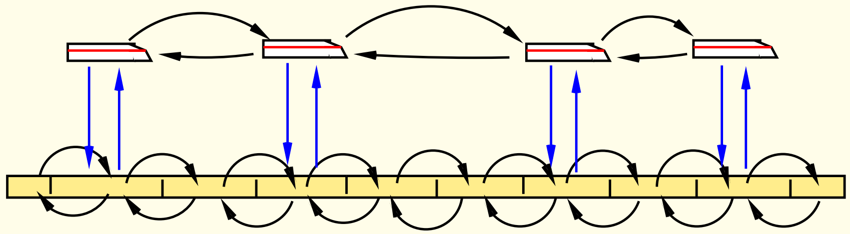
  (2) $\models_{\mathcal{T}_S} \mathsf{Safe}(\overline{x}) \wedge \mathsf{Update}(\overline{x}, \overline{x}') \rightarrow \mathsf{Safe}(\overline{x}')$ (Safe holds before $\Rightarrow$ holds after upda

- **Bounded model checking (BMC):**

  Check whether, for a fixed k, unsafe states are reachable in at most k steps, i.e. for all $0 \leq j \leq k$:

$$\mathsf{Init}(x_0) \wedge \mathsf{Update}_1(x_0, x_1) \wedge \cdots \wedge \mathsf{Update}_n(x_{j-1}, x_j) \wedge \neg \mathsf{Safe}(x_j) \models_{\mathcal{T}_S} \perp$$

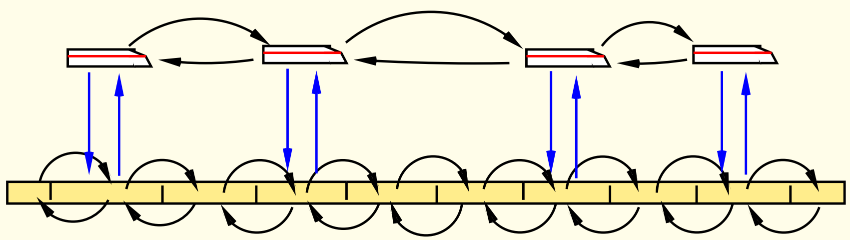
Trains on a linear track



Example 1: Speed Update

$$
\begin{array}{lcl}
\mathrm{pos}(t) < \mathrm{length}(\mathrm{segm}(t)) - d & \rightarrow & 0 \leq \mathrm{spd}'(t) \leq \mathrm{lmax}(\mathrm{segm}(t)) \\
\mathrm{pos}(t) \geq \mathrm{length}(\mathrm{segm}(t)) - d & \wedge & \mathrm{alloc}(\mathrm{next}_s(\mathrm{segm}(t))) = \mathrm{tid}(t) \\
 & \rightarrow & 0 \leq \mathrm{spd}'(t) \leq \min(\mathrm{lmax}(\mathrm{segm}(t)), \mathrm{lmax}(\mathrm{next}_s(\mathrm{segm}(t))) \\
\mathrm{pos}(t) \geq \mathrm{length}(\mathrm{segm}(t)) - d & \wedge & \mathrm{alloc}(\mathrm{next}_s(\mathrm{segm}(t))) \neq \mathrm{tid}(t) \\
 & \rightarrow & \mathrm{spd}'(t) = \max(\mathrm{spd}(t) - \mathrm{decmax}, 0)
\end{array}
$$

Trains on a linear track



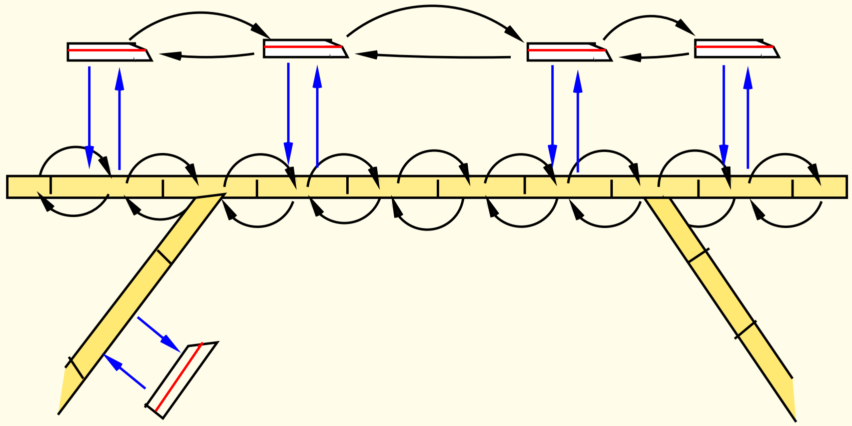
Example 1: Speed Update

$$
\begin{array}{lll}
\mathsf{pos}(t) < \mathsf{length}(\mathsf{segm}(t)) - d & \rightarrow & 0 \leq \mathsf{spd}'(t) \leq \mathsf{lmax}(\mathsf{segm}(t)) \\
\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d & \wedge & \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) = \mathsf{tid}(t) \\
 & \rightarrow & 0 \leq \mathsf{spd}'(t) \leq \mathsf{min}(\mathsf{lmax}(\mathsf{segm}(t)), \mathsf{lmax}(\mathsf{next}_s(\mathsf{segm}(t))) \\
\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d & \wedge & \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) \neq \mathsf{tid}(t) \\
 & \rightarrow & \mathsf{spd}'(t) = \mathsf{max}(\mathsf{spd}(t) - \mathsf{decmax}, 0)
\end{array}
$$

Proof task:

$\mathsf{Safe}(\mathsf{pos}, \mathsf{next}, \mathsf{prev}, \mathsf{spd}) \wedge \mathsf{SpeedUpdate}(\mathsf{pos}, \mathsf{next}, \mathsf{prev}, \mathsf{spd}, \mathsf{spd}') \rightarrow \mathsf{Safe}(\mathsf{pos}', \mathsf{next}, \mathsf{prev}, \mathsf{sp}$

Incoming and outgoing trains



Example 2: Enter Update (also updates for segm', spd', pos', train')

Assume: $s_1 \neq \mathrm{null}_s$, $t_1 \neq \mathrm{null}_t$, $\mathrm{train}(s) \neq t_1$, $\mathrm{alloc}(s_1) = \mathrm{idt}(t_1)$

$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){<}\mathrm{ids}(s_1), \mathrm{next}_t(t){=}\mathrm{null}_t, \mathrm{alloc}(s_1){=}\mathrm{tid}(t_1) \rightarrow \mathrm{next}'(t){=}t_1 \wedge \mathrm{next}'(t_1){=}\mathrm{null}_t$

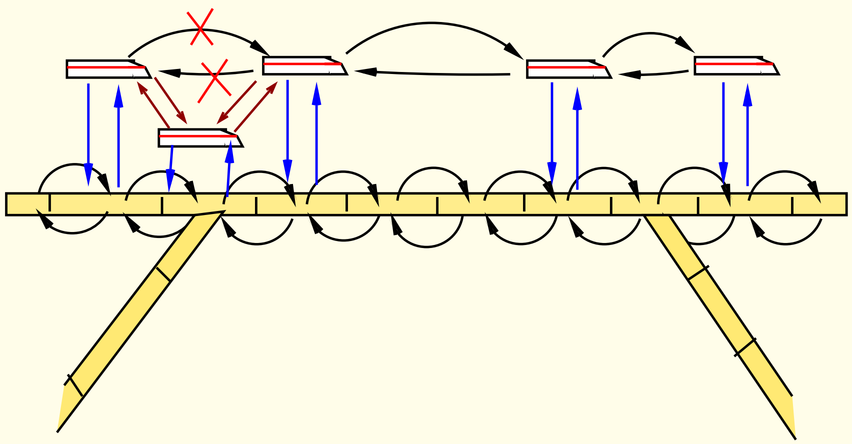
$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){<}\mathrm{ids}(s_1), \mathrm{alloc}(s_1){=}\mathrm{tid}(t_1), \mathrm{next}_t(t){\neq}\mathrm{null}_t, \mathrm{ids}(\mathrm{segm}(\mathrm{next}_t(t))){\leq}\mathrm{ids}(s_1)$

$\quad \rightarrow \mathrm{next}'(t){=}\mathrm{next}_t(t)$

...

$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){\geq}\mathrm{ids}(s_1) \rightarrow \mathrm{next}'(t){=}\mathrm{next}_t(t)$

Incoming and outgoing trains



Example 2: Enter Update (also updates for segm', spd', pos', train')

Assume: $s_1 \neq \mathrm{null}_s$, $t_1 \neq \mathrm{null}_t$, $\mathrm{train}(s) \neq t_1$, $\mathrm{alloc}(s_1) = \mathrm{idt}(t_1)$

$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){<}\mathrm{ids}(s_1), \mathrm{next}_t(t){=}\mathrm{null}_t, \mathrm{alloc}(s_1){=}\mathrm{tid}(t_1) \rightarrow \mathrm{next}'(t){=}t_1 \wedge \mathrm{next}'(t_1){=}\mathrm{null}_t$

$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){<}\mathrm{ids}(s_1), \mathrm{alloc}(s_1){=}\mathrm{tid}(t_1), \mathrm{next}_t(t){\neq}\mathrm{null}_t, \mathrm{ids}(\mathrm{segm}(\mathrm{next}_t(t))){\leq}\mathrm{ids}(s_1)$

$\rightarrow \mathrm{next}'(t){=}\mathrm{next}_t(t)$

...

$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){\geq}\mathrm{ids}(s_1) \rightarrow \mathrm{next}'(t){=}\mathrm{next}_t(t)$

Safety property

Safety property we want to prove:
no two different trains ever occupy the same track segment:

$$(\text{Safe}) \quad \forall t_1, t_2 \;\; \text{segm}(t_1) = \text{segm}(t2) \rightarrow t_1 = t_2$$

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant (Inv_i) for every control location i of the TCS, and prove:

(1) $(\text{Inv}_i) \models (\text{Safe})$ for all locations i and

(2) the invariants are preserved under all transitions of the system,

$(\text{Inv}_i) \wedge (\text{Update}) \models (\text{Inv}'_j)$

whenever (Update) is a transition from location i to j .

Safety property

Safety property we want to prove:
no two different trains ever occupy the same track segment:

$$(\mathsf{Safe}) \quad \forall t_1, t_2 \;\; \mathsf{segm}(t_1) = \mathsf{segm}(t2) \rightarrow t_1 = t_2$$

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant (Inv_i) for every control location i of the TCS, and prove:

(1) (Inv_i) $\models$ (Safe) for all locations i and

(2) the invariants are preserved under all transitions of the system,
$(\mathsf{Inv}_i) \wedge (\mathsf{Update}) \models (\mathsf{Inv}'_j)$
whenever (Update) is a transition from location i to j .

Here: Inv_i generated by hand (use poss. of generating counterexamples with H-PILoT)

Verification problems

(1) $(\mathsf{Inv}_i) \models (\mathsf{Safe})$ for all locations i and

(2) the invariants are preserved under all transitions of the system,

$(\mathsf{Inv}_i) \wedge (\mathsf{Update}) \models (\mathsf{Inv}'_j)$

whenever (Update) is a transition from location i to j .

Ground satisfiability problems for pointer data structures

Problem: Axioms, Invariants: are universally quantified

Our solution: Hierarchical reasoning in local theory extensions

Modularity in automated reasoning

Examples of theories we need to handle

- **Invariants**

$$(\mathsf{Inv}_1)\ \forall t : \mathsf{Train}.\ \mathsf{pc} \neq \mathsf{InitState} \wedge \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) \neq \mathsf{tid}(t)$$
$$\rightarrow \mathsf{length}(\mathsf{segm}(t)) - \mathsf{bd}(\mathsf{spd}(t)) > \mathsf{pos}(t) + \mathsf{spd}(t) \cdot \Delta t$$

$$(\mathsf{Inv}_2)\ \forall t : \mathsf{Train}.\ \mathsf{pc} \neq \mathsf{InitState} \wedge \mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d$$
$$\rightarrow \mathsf{spd}(t) \leq \mathsf{lmax}(\mathsf{next}_s(\mathsf{segm}(t)))$$

Modularity in automated reasoning

Examples of theories we need to handle

- **Invariants**

$$(\mathsf{Inv}_1)\ \forall t : \mathsf{Train}.\ \mathsf{pc} \neq \mathsf{InitState} \wedge \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) \neq \mathsf{tid}(t)$$
$$\rightarrow \mathsf{length}(\mathsf{segm}(t)) - \mathsf{bd}(\mathsf{spd}(t)) > \mathsf{pos}(t) + \mathsf{spd}(t) \cdot \Delta t$$
$$(\mathsf{Inv}_2)\ \forall t : \mathsf{Train}.\ \mathsf{pc} \neq \mathsf{InitState} \wedge \mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d$$
$$\rightarrow \mathsf{spd}(t) \leq \mathsf{lmax}(\mathsf{next}_s(\mathsf{segm}(t)))$$

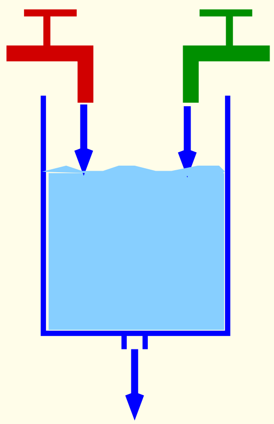
- **Update rules**

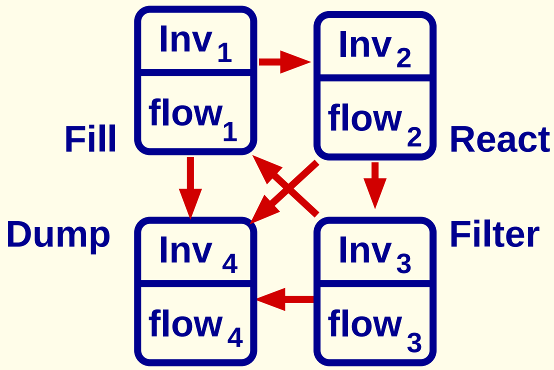
$$\forall t : \phi_1(t) \quad \rightarrow \quad s_1 \leq \mathsf{spd}'(t) \leq t_1$$
$$\dots$$
$$\forall t : \phi_n(t) \quad \rightarrow \quad s_n \leq \mathsf{spd}'(t) \leq t_n$$

Example 2

Hybrid systems $\mapsto$ Hybrid automata

Example 2





Chemical plant

Two substances are mixed; they react. The resulting product is filtered out; then the procedure is repeated.

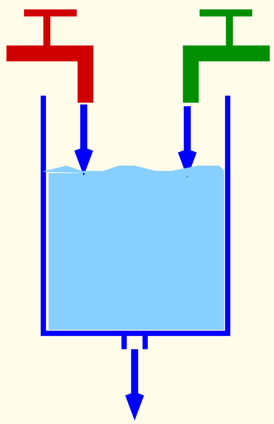
Check:

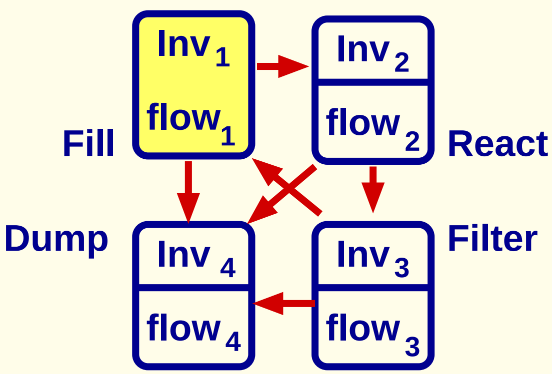
- No overflow
- Substances always in the right proportion
- If substances in wrong proportion,
  tank can be drained in $\leq$ 200s.

Parametric description:

- Determine values for parameters
  such that this is the case

Example 2





Mode 1: Fill Temperature is low, 1 and 2 do not react.

Substances 1 and 2 (possibly mixed with a small quantity of 3) are filled in the tank in equal quantities up to a margin of error.

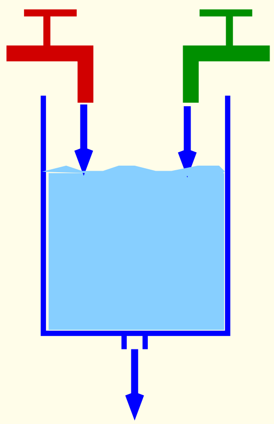
Inv_1	$x_1 + x_2 + x_3 \leq L_f \ \wedge \ \bigwedge_{i=1}^{3} x_i \geq 0 \ \wedge$ $-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \ \wedge \ 0 \leq x_3 \leq \mathsf{min}$
flow_1	$\dot{x_1} \geq \mathsf{dmin} \wedge \dot{x_2} \geq \mathsf{dmin} \wedge \dot{x_3} = 0 \wedge -\delta_a \leq \dot{x_1} - \dot{x_2} \leq \delta_a$

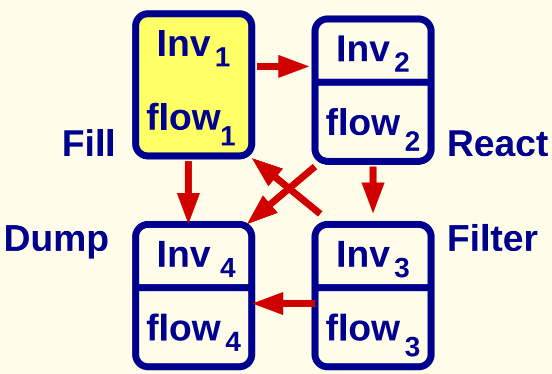
Jumps: (1,4)

If proportion not kept: system jumps into mode 4 (**Dump**)

e_1	$\mathsf{guard}_{e_1}(x_1, x_2, x_3) = x_1 - x_2 \geq \epsilon_a$
(from 1 to 4)	$\mathsf{jump}_{e_1}(x_1, x_2, x_3, x_1', x_2', x_3') = \bigwedge_{i=1}^{3} x_i' = 0$
e_2	$\mathsf{guard}_{e_1}(x_1, x_2, x_3) = x_1 - x_2 \leq -\epsilon_a$
(from 1 to 4)	$\mathsf{jump}_{e_1}(x_1, x_2, x_3, x_1', x_2', x_3') = \bigwedge_{i=1}^{3} x_i' = 0$

Example





Mode 1: Fill Temperature is low, 1 and 2 do not react.

Substances 1 and 2 (possibly mixed with a small quantity of 3) are filled in the tank in equal quantities up to a margin of error.

$$\mathrm{Inv}_1 \qquad x_1 + x_2 + x_3 \leq L_f \;\wedge\; \bigwedge_{i=1}^{3} x_i \geq 0 \;\wedge$$
$$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \;\wedge\; 0 \leq x_3 \leq \mathrm{min}$$

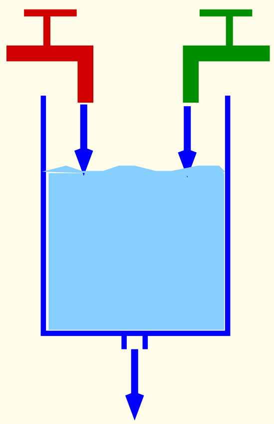
$$\mathrm{flow}_1 \qquad \dot{x_1} \geq \mathrm{dmin} \wedge \dot{x_2} \geq \mathrm{dmin} \wedge \dot{x_3} = 0 \wedge -\delta_a \leq \dot{x_1} - \dot{x_2} \leq \delta_a$$

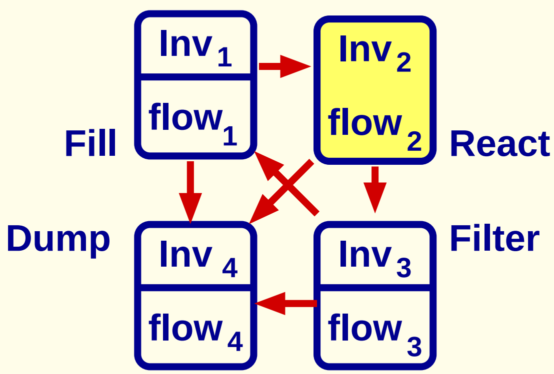
Jumps: (1,2)

If the total quantity of substances exceeds level L_f (tank filled) the system jumps into mode 2 (**React**).

$$e = (1,2) \qquad \mathrm{guard}_{(1,2)}(x_1, x_2, x_3) = x_1 + x_2 + x_3 \geq L_f$$
$$\mathrm{jump}_{(1,2)}(x_1, x_2, x_3, x_1', x_2', x_3') = \bigwedge_{i=1}^{3} x_i' = x_i$$

Example





Mode 2: React Temperature is high. Substances 1 and 2 react. The reaction consumes equal quantities of substances 1 and 2 and produces substance 3.

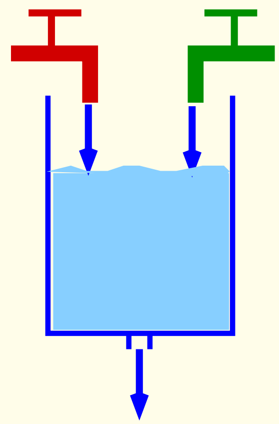
$$
\begin{array}{ll}
\mathrm{Inv}_2 & L_f \le x_1 + x_2 + x_3 \le L_{\mathrm{overflow}} \;\wedge\; \bigwedge_{i=1}^{3} x_i \ge 0 \;\wedge \\
& -\epsilon_a \le x_1 - x_2 \le \epsilon_a \;\wedge\; 0 \le x_3 \le \mathrm{max} \\
& \\
\mathrm{flow}_2 & \dot{x}_1 \le -\mathrm{dmax} \wedge \dot{x}_2 \le -\mathrm{dmax} \wedge \dot{x}_3 \ge \mathrm{dmin} \\
& \wedge \dot{x}_1 = \dot{x}_2 \wedge \dot{x}_3 + \dot{x}_1 + \dot{x}_2 = 0
\end{array}
$$

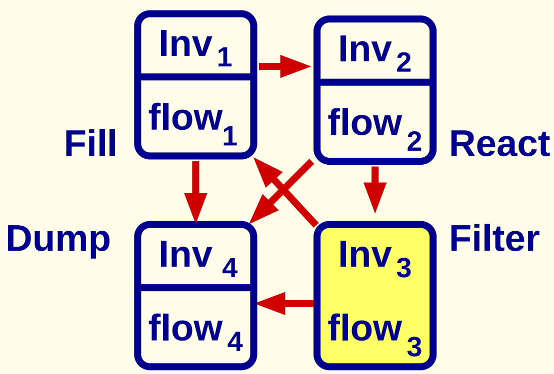
Jumps:

If the proportion between substances 1 and 2 is not kept the system jumps into mode 4 (**Dump**);

If the total quantity of substances 1 and 2 is below some minima level min the system jumps into mode 3 (**Filter**).

Example





Mode 3: Filter Temperature is low. Substance 3 is filtered out.

$$\mathrm{Inv}_3 \qquad x_1 + x_2 + x_3 \leq L_{\mathrm{overflow}} \;\wedge\; \bigwedge_{i=1}^{3} x_i \geq 0 \;\wedge$$
$$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \;\wedge\; x_3 \geq \mathrm{min}$$

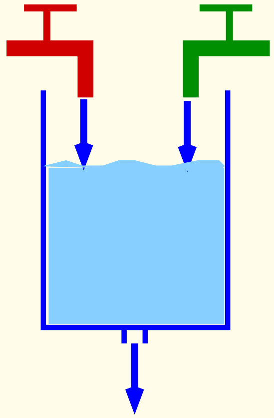
$$\mathrm{flow}_3 \qquad \dot{x}_1 = 0 \wedge \dot{x}_2 = 0 \;\wedge\; \dot{x}_3 \leq -\mathrm{dmax}$$

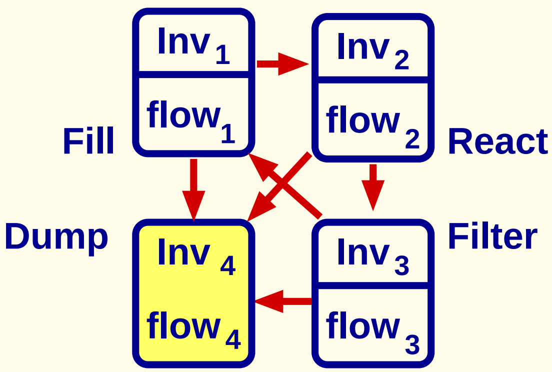
Jumps:

If proportion not kept: system jumps into mode 4 (**Dump**);

Otherwise, if the concentration of substance 3 is below some minimal level min the system jumps into mode 1 (**Fill**).

Example





Mode 4: Dump The content of the tank is emptied.

For simplicity we assume that this happens instantaneously:

$$\mathrm{Inv}_4 : \bigwedge_{i=1}^{3} x_i = 0 \text{ and } \mathrm{flow}_4 : \bigwedge_{i=1}^{3} \dot{x}_i = 0.$$

Simple verification problems

Invariant checking: Check whether Ψ is an invariant in a HA S, i.e.:

(1) $\text{Init}_q \models \Psi$ for all $q \in Q$;

(2) Ψ is invariant under jumps and flows:

(Flow) For every flow in mode q, the continuous variables satisfy Ψ during and at the end of the flow.

(Jump) For every jump according to a control switch e, if Ψ holds before the jump, it holds after the jump.

Examples:

- Is "$x_1 + x_2 + x_3 \leq L_{\text{overflow}}$" an invariant? (no overflow)
- Is "$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a$" an invariant?
  (substances always mixed in the right proportion)

Simple verification problems

Bounded model checking: Is formula Safe preserved under runs of length $\leq k$?, i.e.:

(1) $\text{Init}_q \models \text{Safe}$ for every $q \in Q$;

(2) The continuous variables satisfy Safe during and at the end of all runs of length j for all $1 \leq j \leq k$.

Example:

- Is "$x_1 + x_2 + x_3 \leq L_{\text{overflow}}$" true after all runs of length $\leq k$ starting from a state with e.g. $x_1 = x_2 = x_3 = 0$?

- Is "$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a$" true after all runs of length $\leq k$ starting from a state with $x_1 = x_2 = x_3 = 0$?

Simple verification problems

Reductions of verification problems to linear arithmetic

(1) Mode invariants, initial states and guards of mode switches are described as conjunctions of linear inequalities.

Example: $\mathsf{Inv}_q = \bigwedge_{j=1}^{m_q} (\sum_{i=1}^{n} a_{ij}^q x_i \leq a_j^q)$ can be expressed by:

$$\mathsf{Inv}_q(x_1(t), \dots, x_n(t)) = \bigwedge_{j=1}^{m_q} (\sum_{i=1}^{n} a_{ij}^q x_i(t) \leq a_j^q)$$

Simple verification problems

Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:

$\text{flow}_q = \bigwedge_{j=1}^{n_q}(\sum_{i=1}^{n} c_{ij}^{q} \dot{x}_i \leq c_j^q)$, i.e. $\text{flow}_q(t) = \bigwedge_{j=1}^{n_q}(\sum_{i=1}^{n} c_{ij}^{q} \dot{x}_i(t) \leq c_j^q)$.

Simple verification problems

Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:

$\mathrm{flow}_q = \bigwedge_{j=1}^{n_q} (\sum_{i=1}^{n} c_{ij}^q \dot{x}_i \leq c_j^q)$, i.e. $\mathrm{flow}_q(t) = \bigwedge_{j=1}^{n_q} (\sum_{i=1}^{n} c_{ij}^q \dot{x}_i(t) \leq c_j^q)$.

Approach: Express the flow conditions in $[t_0, t_1]$ without referring to derivatives.

$\mathrm{Flow}_q(t_0, t_1) : \forall t (t_0 \leq t \leq t_1 \rightarrow \mathrm{Inv}_q(\overline{x}(t))) \ \wedge \ \forall t, t' (t_0 \leq t \leq t' \leq t_1 \rightarrow \underline{\mathrm{flow}}_q(t, t'))$.

where: $\underline{\mathrm{flow}}_q(t, t') = \bigwedge_{j=1}^{n_q} (\sum_{i=1}^{n} c_{ij}^q (x_i(t') - x_i(t)) \leq c_j^q (t' - t))$.

Simple verification problems

Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:

$\mathsf{flow}_q = \bigwedge_{j=1}^{n_q}(\sum_{i=1}^{n} c_{ij}^q \dot{x}_i \leq c_j^q)$, i.e. $\mathsf{flow}_q(t) = \bigwedge_{j=1}^{n_q}(\sum_{i=1}^{n} c_{ij}^q \dot{x}_i(t) \leq c_j^q)$.

Approach: Express the flow conditions in $[t_0, t_1]$ without referring to derivatives.

$\mathsf{Flow}_q(t_0, t_1) : \forall t (t_0 \leq t \leq t_1 \rightarrow \mathsf{Inv}_q(\overline{x}(t))) \; \wedge \; \forall t, t' (t_0 \leq t \leq t' \leq t_1 \rightarrow \underline{\mathsf{flow}}_q(t, t'))$.

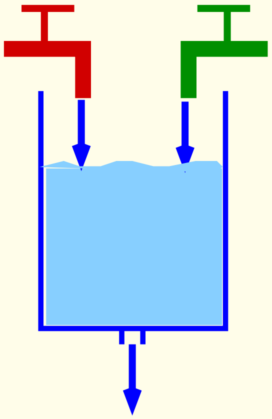
where: $\underline{\mathsf{flow}}_q(t, t') = \bigwedge_{j=1}^{n_q}(\sum_{i=1}^{n} c_{ij}^q (x_i(t') - x_i(t)) \leq c_j^q (t' - t))$.

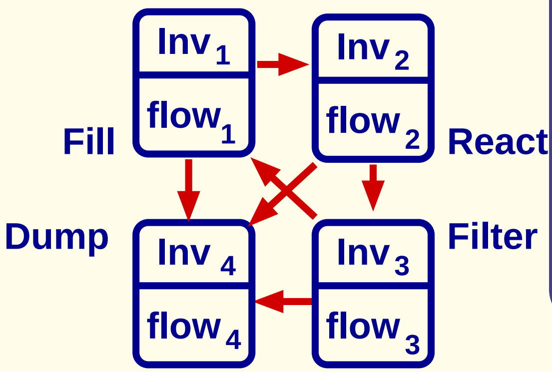
Remark: $\mathsf{Flow}_q(t_0, t_1)$ contains universal quantifiers.

Locality results: Sufficient to use the instances at t_0 and t_1

$\mathsf{Flow}_q^{\mathsf{Inst}}(t_0, t_1) : \mathsf{Inv}_q(\overline{x}(t_0))) \; \wedge \; \mathsf{Inv}_q(\overline{x}(t_1))) \; \wedge \; \underline{\mathsf{flow}}_q(t_0, t_1))$.

Example





Invariant:

$\phi_{\text{safe}}(x_1, x_2, x_3) : x_1 + x_2 + x_3 \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1 - x_2 \leq \epsilon.$

Illustration: $F_{\text{flow}}(2)$ (invariance under the flow in reaction mode):

$$\begin{array}{r|l}
\Psi(0) & (x_1(0)+x_2(0)+x_3(0) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(0)-x_2(0) \leq \epsilon) \wedge \\
\neg\Psi(t) & \neg(x_1(t)+x_2(t)+x_3(t) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(t)-x_2(t) \leq \epsilon) \wedge
\end{array}$$

$$\begin{array}{r|l}
\mathrm{Inv}_2(0) & L_f \leq x_1(0) + x_2(0) + x_3(0) \leq L_{\text{overflow}} \wedge x_3(0) \leq \max \wedge \\
\mathrm{Inv}_2(t) & L_f \leq x_1(t) + x_2(t) + x_3(t) \leq L_{\text{overflow}} \wedge x_3(t) \leq \max \wedge
\end{array}$$

$$\begin{array}{r|l}
\mathrm{flow}_2 & x_1(t)-x_1(0) \leq -\mathrm{dmax}\cdot t \ \wedge \ x_2(t)-x_2(0) \leq -\mathrm{dmax}\cdot t \ \wedge \\
 & x_3(t)-x_3(0) \geq \mathrm{dmin}\cdot t \wedge (x_1(t)-x_1(0))-(x_2(t)-x_2(0)) = 0 \ \wedge \\
 & (x_1(t)-x_1(0))+(x_2(t)-x_2(0))+(x_3(t)-x_3(0)) = 0
\end{array}$$

For fixed values for L_f, L_{overflow} – satisfiability check: PTIME.

Parametric version: check satisfiability if $L_f < L_{\text{overflow}} \wedge \epsilon_a < \epsilon$
or generate constraints on the parameters which guarantee (un)satisfiability

Other approaches

First-Order Dynamic Logic

Dynamic logic in which the atomic programs contain variables

The KeY System (Bernhard Beckert et al.)

Hybrid Dynamic Logic

Dynamic logic in which the atomic programs contain differential equations

The KeYmaera Verification Tool (Andre Platzer)

(Differential dynamic logic)