

# Formal Specification and Verification

Deductive Verification: An introduction (2)

14.02.2017

Viorica Sofronie-Stokkermans

e-mail: [sofronie@uni-koblenz.de](mailto: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:

$(l, \beta)$ , where  $l$  location and  $\beta$  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  $(l, \beta)$  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:**  $(l, \beta) \rightarrow (l', \beta')$

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

**Example:**

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

**Abbreviations:**

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

$\text{skip}(v_1, \dots, v_n) := (v'_1 = v_1 \wedge \dots \wedge v'_n = v_n)$

# Programs as transition systems

---

**Verification problem: Program + Description of the “bad” states**

**Succinct representation:**

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

- $V$  - 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  $V'$

- $\phi_{err}$  - 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:  $y \geq z \models y + 1 \geq z$

# 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  $Init := (pc = l_1)$
- Error condition  $\phi_{\text{err}} := (pc = l_5)$
- Program transitions  $\mathcal{R} = \{\rho_1, \dots, \rho_5\}$ , where:
  - $\rho_1 = (\text{move}(l_1, l_2) \wedge y \geq z \wedge \text{skip}(x, y, z))$
  - $\rho_2 = (\text{move}(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge \text{skip}(y, z))$
  - $\rho_3 = (\text{move}(l_2, l_3) \wedge x \geq y \wedge \text{skip}(x, y, z))$
  - $\rho_4 = (\text{move}(l_3, l_4) \wedge x \geq z \wedge \text{skip}(x, y, z))$
  - $\rho_5 = (\text{move}(l_3, l_5) \wedge x + 1 \leq z \wedge \text{skip}(x, y, z))$

# 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  $(s, s') \models \rho_{\mathcal{R}}$ :
  - $s : V \rightarrow \bigcup_{x \in V} D_x$ ,  $s(x) \in D_x$  for all  $x \in V$
  - $s' : V' \rightarrow \bigcup_{x \in V} D_x$ ,  $s'(x') \in D_x$  for all  $x \in V$
  - $\beta : V \cup V' \rightarrow \bigcup_{x \in X} D_x$  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  $s_1 s_2 \dots$  such that:

- The first element is an initial state, i.e.,  $s_1 \models \textit{Init}$
- 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  $\phi_{reach}$ ,

$$\phi_{err} \wedge \phi_{reach} \models \perp$$

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

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

$$Init \wedge \phi_{reach-1}(\phi_{err}) \models \perp$$

where  $\phi_{reach-1}(\phi_{err})$  = set of program states from which some state in  $\phi_{err}$  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:

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

# Post operator

---

Let  $\phi$  be a formula over  $V$

Let  $\rho$  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  $\phi$  under the relation  $\rho$ .

# Post operator

---

Let  $\phi$  be a formula over  $V$

Let  $\rho$  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  $\phi$  under the relation  $\rho$ .

*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  $\rho := \rho_2$

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

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

# Application of post in example program

---

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

Transition relation  $\rho := \rho_2$

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

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

[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  $\phi$  under  $\rho$

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

Characterize  $\phi_{\text{reach}}$  using iterates of  $post$ :

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

disjuncts = iterates for every natural number  $n$  (“ $\omega$ -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  $\phi$  be a formula over  $V$

Let  $\rho$  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  $\phi$  under the relation  $\rho$ .

Computation of  $pre^n$  similar.

# Problem

---

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

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

i.e. fixpoint reached.

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

**How to check whether error states are reachable?**

$\phi_{\text{reach}}, \phi_{\text{err}}$  are formulae.

No error states are reachable iff  $\phi_{\text{reach}} \wedge \phi_{\text{err}} \models \perp$

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

⇒ Decision Procedures for Verification

# 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

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

## Backjump

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

# 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  $\mathcal{T} = \text{UIF}$  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

# 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**

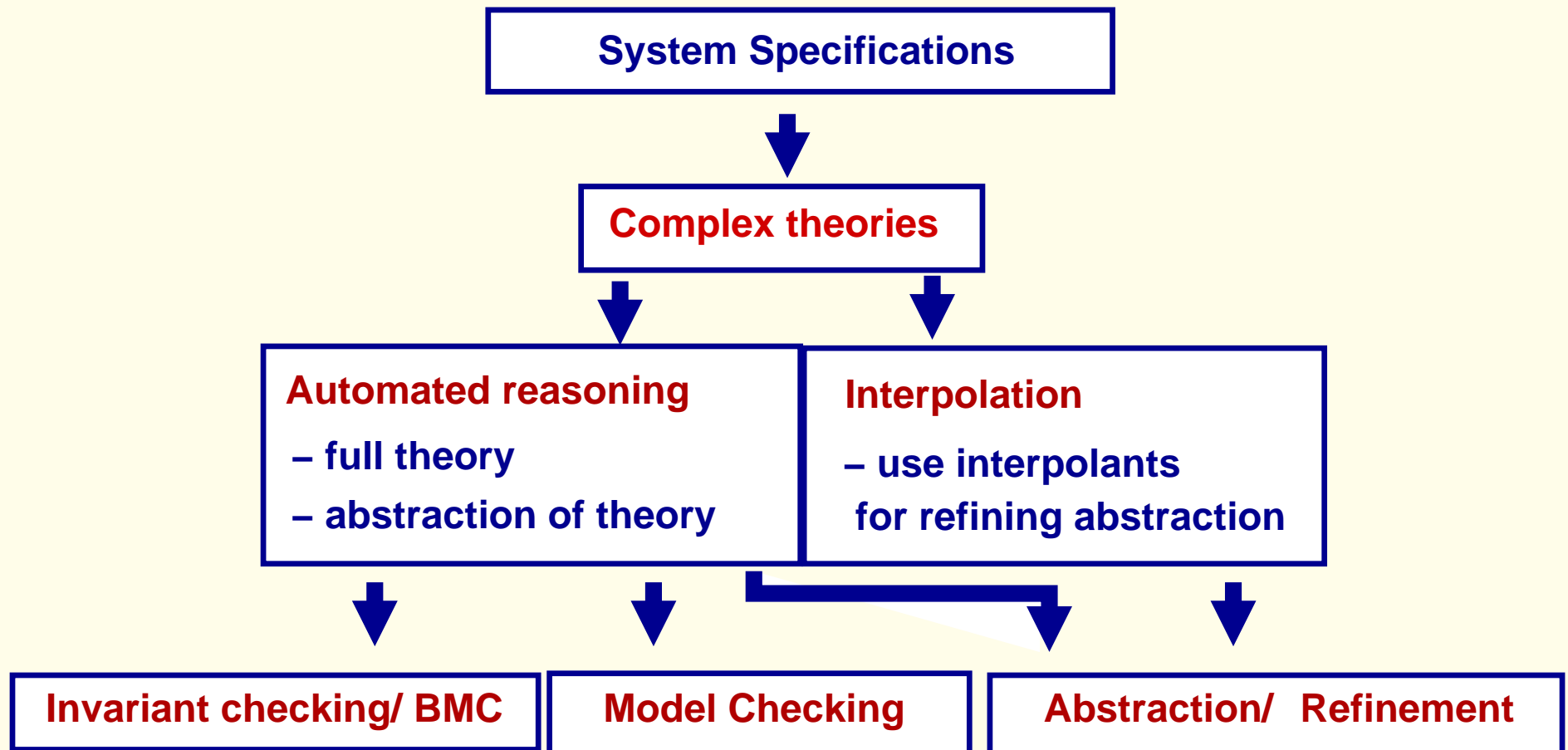
**Model Checking**

**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  
 $\text{Init}(\bar{x}); \rho_{\mathcal{R}}(\bar{x}, \bar{x}')$

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

- **Invariant checking**

- (1)  $\mathcal{T}_S \models \text{Init}(\bar{x}) \rightarrow \text{Safe}(\bar{x})$  (Safe holds in the initial state)
- (2)  $\mathcal{T}_S \models \text{Safe}(\bar{x}) \wedge \rho_{\mathcal{R}}(\bar{x}, \bar{x}') \rightarrow \text{Safe}(\bar{x}')$  (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  $0 \leq j \leq k$ :

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

# 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

---

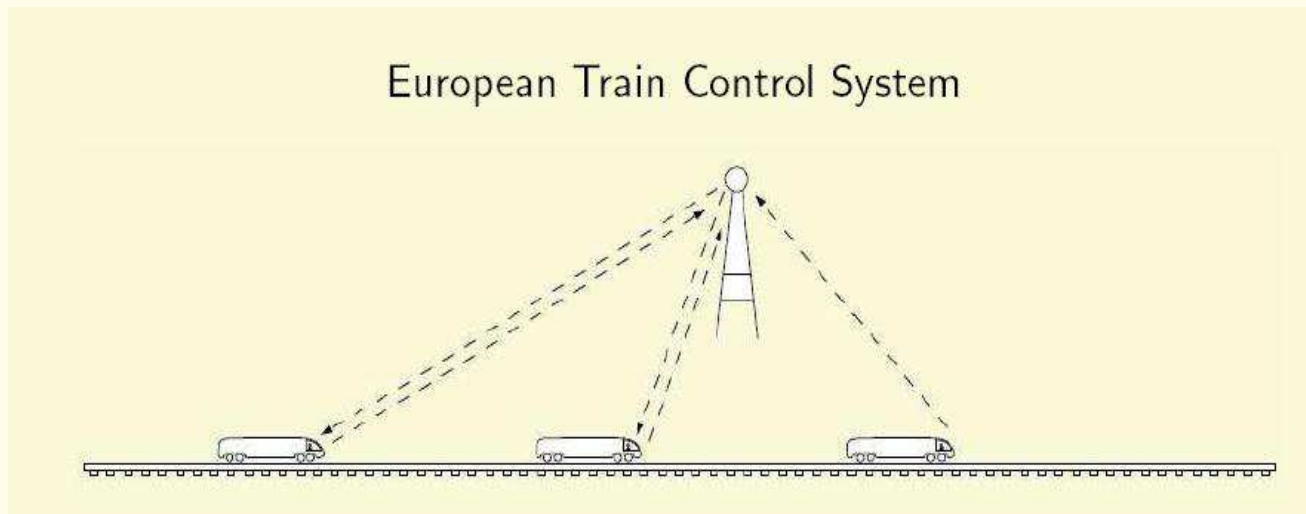
- First order logic is undecidable
- In applications, theories do not occur alone
  - ⇒ need to consider combinations of theories
- + Fragments of theories occurring in applications are often decidable
- + Often provers for the component theories can be combined efficiently

## 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)

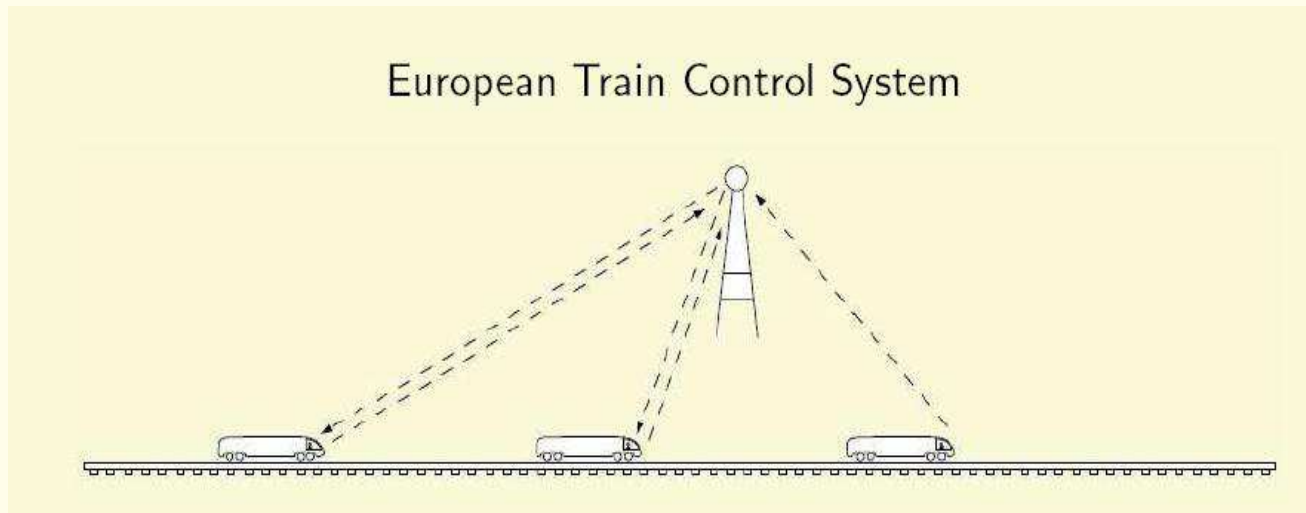
Simplified version of ETCS Case Study [Jacobs,VS'06, Faber,Jacobs,VS'07]



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

# Example: ETCS Case Study (AVACS project)

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



Update(**pos**, **pos'**) :

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

...

# Example: ETCS Case Study (AVACS project)

---

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

**Inductive invariant:**       $\text{Safe}(\text{pos}) \wedge \text{Update}(\text{pos}, \text{pos}') \wedge \neg \text{Safe}(\text{pos}') \models_{\mathcal{T}_S} \perp$

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

**Problem:** Satisfiability test for quantified formulae in complex theory

# More complex ETCS Case studies

---

[Faber, Jacobs, VS, 2007]

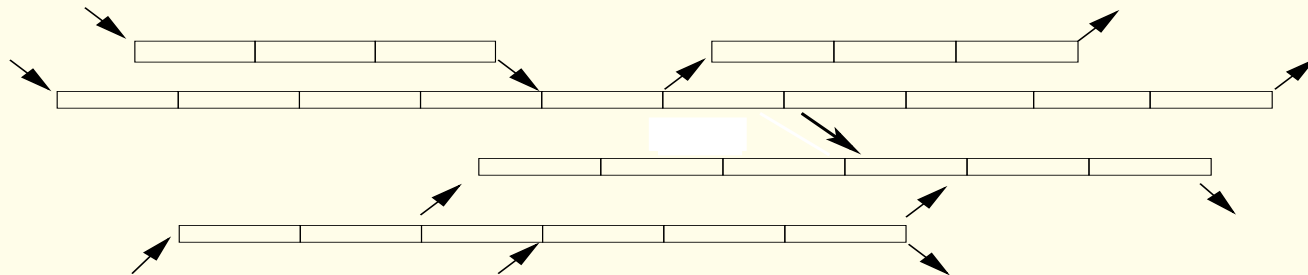
- Take into account also:
  - Emergency messages
  - Durations
- Specification language: CSP-OZ-DC
  - Reduction to satisfiability in theories for which decision procedures exist
- **Tool chain:** [Faber, Ihlemann, Jacobs, VS]  
CSP-OZ-DC  $\mapsto$  Transition constr.  $\mapsto$  Decision procedures (H-PILoT)



## Example 2: Parametric topology

---

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



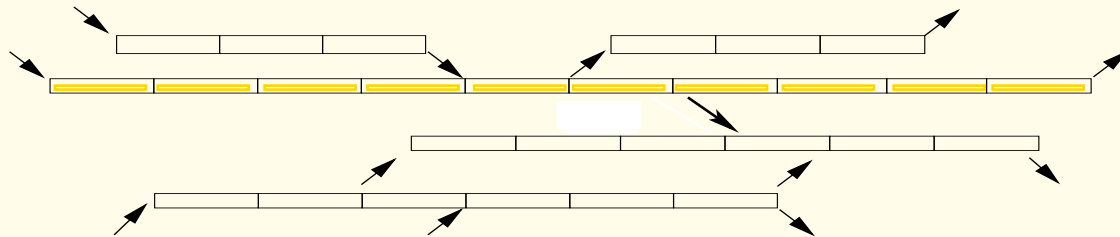
### Assumptions:

- 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:

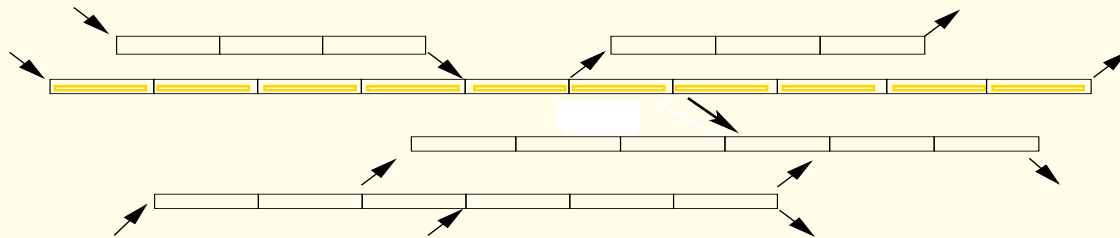
- No cycles
- in-degree (out-degree) of associated graph at most 2.

## 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:

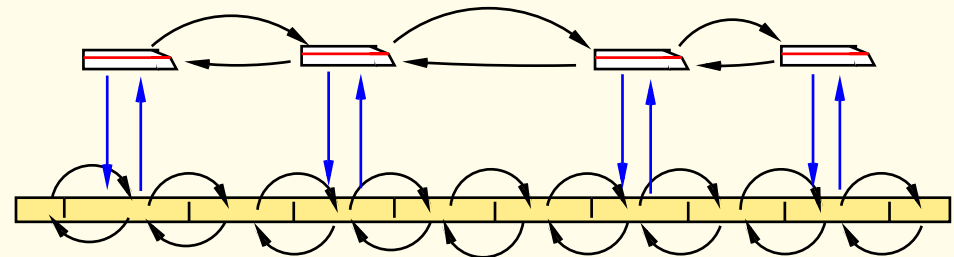
- No cycles
- in-degree (out-degree) of associated graph at most 2.

## Data structures:

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

$p_1$ : trains

$p_2$ : segments



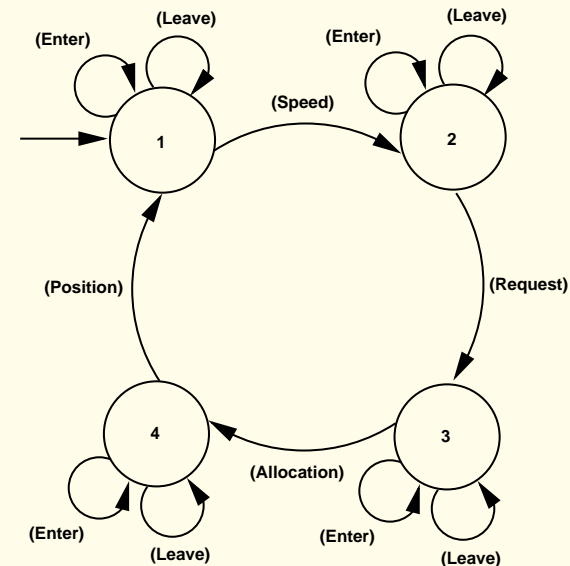
# Example: Controller for line track (RBC)

---

**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)

---

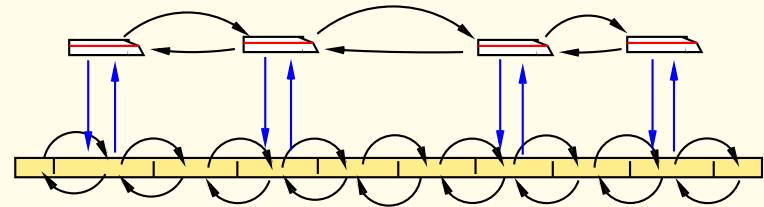
**OZ part.** Consists of data classes, axioms, the Init formulae, update rules.

- **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
  - $gmax : \mathbb{R}$  (the global maximum speed),
  - $decmax : \mathbb{R}$  (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 $\text{Init}(\bar{x}); \text{Update}(\bar{x}, \bar{x}')$
------------------------	--

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

- **Invariant checking**

- (1)  $\models_{\mathcal{T}_S} \text{Init}(\bar{x}) \rightarrow \text{Safe}(\bar{x})$  (Safe holds in the initial state)
- (2)  $\models_{\mathcal{T}_S} \text{Safe}(\bar{x}) \wedge \text{Update}(\bar{x}, \bar{x}') \rightarrow \text{Safe}(\bar{x}')$  (Safe holds before  $\Rightarrow$  holds after update)

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

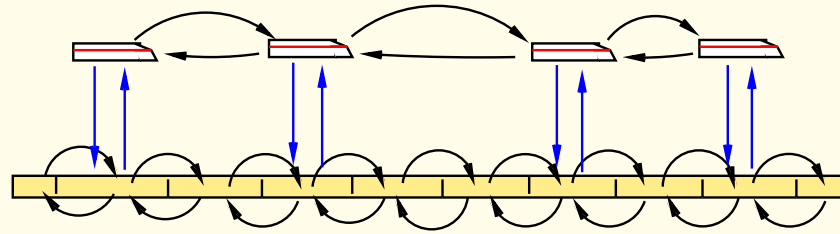
Check whether, for a fixed  $k$ , unsafe states are reachable in at most  $k$  steps, i.e. for all  $0 \leq j \leq k$ :

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



# Trains on a linear track

---



## Example 1: Speed Update

$\text{pos}(t) < \text{length}(\text{segm}(t)) - d \rightarrow 0 \leq \text{spd}'(t) \leq \text{lmax}(\text{segm}(t))$

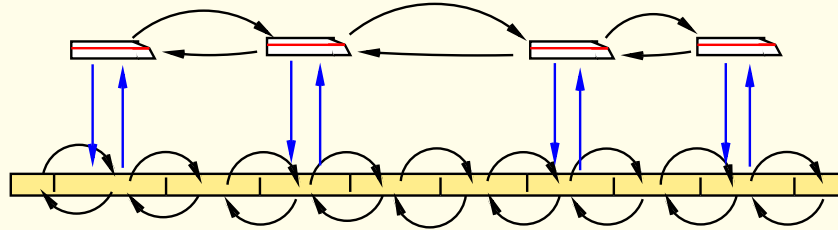
$\text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \wedge \text{alloc}(\text{next}_s(\text{segm}(t))) = \text{tid}(t)$

$\rightarrow 0 \leq \text{spd}'(t) \leq \min(\text{lmax}(\text{segm}(t)), \text{lmax}(\text{next}_s(\text{segm}(t))))$

$\text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \wedge \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t)$

$\rightarrow \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0)$

# Trains on a linear track



## Example 1: Speed Update

$$\text{pos}(t) < \text{length}(\text{segm}(t)) - d \rightarrow 0 \leq \text{spd}'(t) \leq \text{lmax}(\text{segm}(t))$$

$$\text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \wedge \text{alloc}(\text{next}_s(\text{segm}(t))) = \text{tid}(t)$$

$$\rightarrow 0 \leq \text{spd}'(t) \leq \min(\text{lmax}(\text{segm}(t)), \text{lmax}(\text{next}_s(\text{segm}(t))))$$

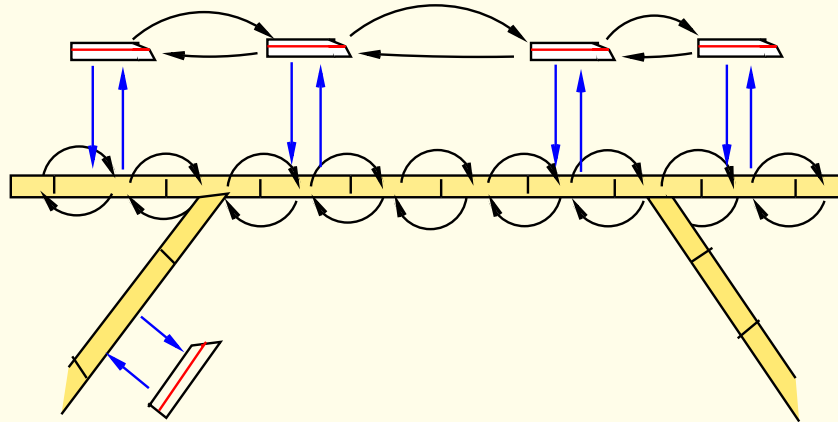
$$\text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \wedge \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t)$$

$$\rightarrow \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0)$$

## Proof task:

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

# Incoming and outgoing trains



**Example 2:** Enter Update (also updates for  $\text{segm}'$ ,  $\text{spd}'$ ,  $\text{pos}'$ ,  $\text{train}'$ )

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

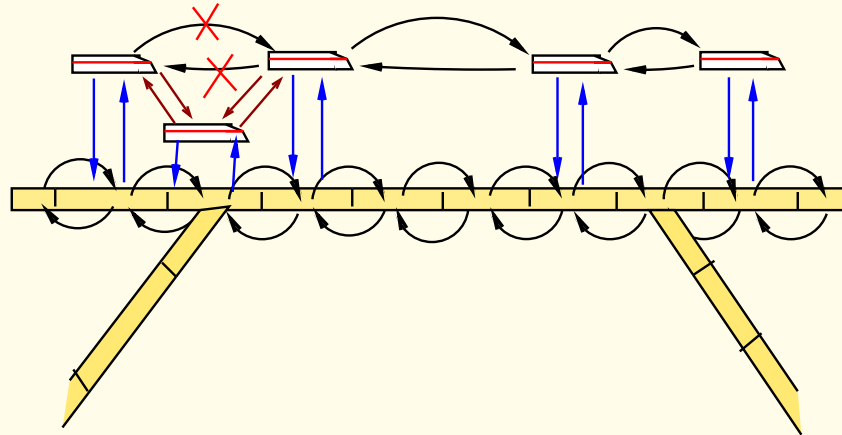
$t \neq t_1$ ,  $\text{ids}(\text{segm}(t)) < \text{ids}(s_1)$ ,  $\text{next}_t(t) = \text{null}_t$ ,  $\text{alloc}(s_1) = \text{tid}(t_1) \rightarrow \text{next}'(t) = t_1 \wedge \text{next}'(t_1) = \text{null}_t$

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

...

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

# Incoming and outgoing trains



**Example 2:** Enter Update (also updates for  $\text{segm}'$ ,  $\text{spd}'$ ,  $\text{pos}'$ ,  $\text{train}'$ )

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

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

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

...

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

# Safety property

---

## **Safety property we want to prove:**

no two different trains ever occupy the same track segment:

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

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant  $(\text{Inv}_i)$  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:

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

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant  $(\text{Inv}_i)$  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  $j$ .

**Here:**  $\text{Inv}_i$  generated by hand (use poss. of generating counterexamples with H-PILoT)

# Verification problems

---

- (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  $(\text{Update})$  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**

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



# Modularity in automated reasoning

---

## Examples of theories we need to handle

- **Invariants**

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

- **Update rules**

$$\forall t : \phi_1(t) \quad \rightarrow \quad s_1 \leq \text{spd}'(t) \leq t_1$$

...

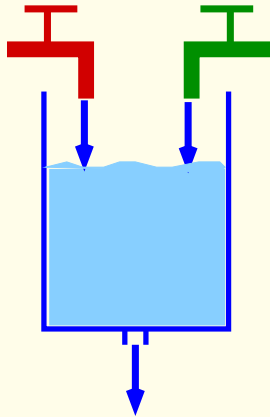
$$\forall t : \phi_n(t) \quad \rightarrow \quad s_n \leq \text{spd}'(t) \leq t_n$$

## Example 2

---

Hybrid systems  $\mapsto$  Hybrid automata

## Example 2

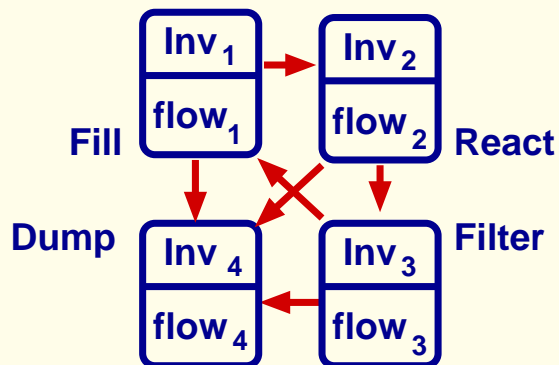


### Chemical plant

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

### Check:

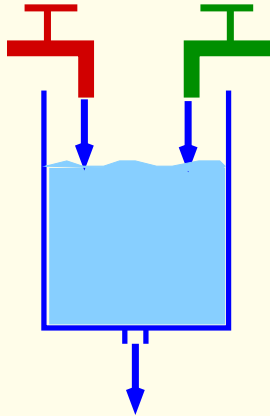
- No overflow
- Substances always in the right proportion
- If substances in wrong proportion, tank can be drained in  $\leq 200$ s.



### Parametric description:

- Determine values for parameters such that this is the case

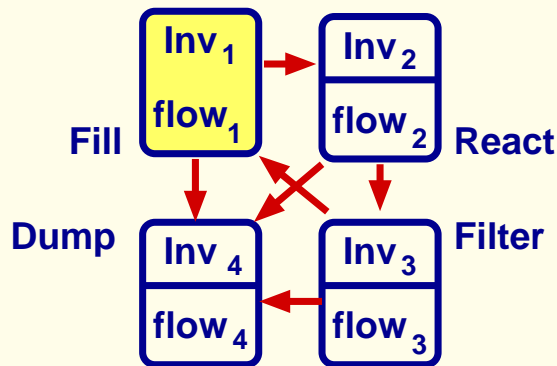
## Example 2



**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.

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

$$\text{flow}_1 \quad \dot{x}_1 \geq \text{dmin} \wedge \dot{x}_2 \geq \text{dmin} \wedge \dot{x}_3 = 0 \wedge -\delta_a \leq \dot{x}_1 - \dot{x}_2 \leq \delta_a$$



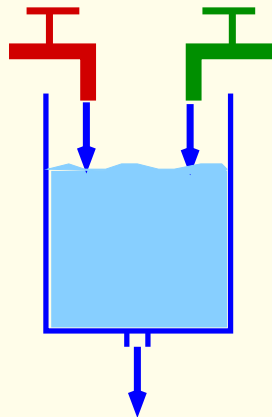
**Jumps: (1,4)**

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

$$\begin{aligned} e_1 \quad & \text{guard}_{e_1}(x_1, x_2, x_3) = x_1 - x_2 \geq \epsilon_a \\ (\text{from 1 to 4}) \quad & \text{jump}_{e_1}(x_1, x_2, x_3, x'_1, x'_2, x'_3) = \bigwedge_{i=1}^3 x'_i = 0 \end{aligned}$$

$$\begin{aligned} e_2 \quad & \text{guard}_{e_1}(x_1, x_2, x_3) = x_1 - x_2 \leq -\epsilon_a \\ (\text{from 1 to 4}) \quad & \text{jump}_{e_1}(x_1, x_2, x_3, x'_1, x'_2, x'_3) = \bigwedge_{i=1}^3 x'_i = 0 \end{aligned}$$

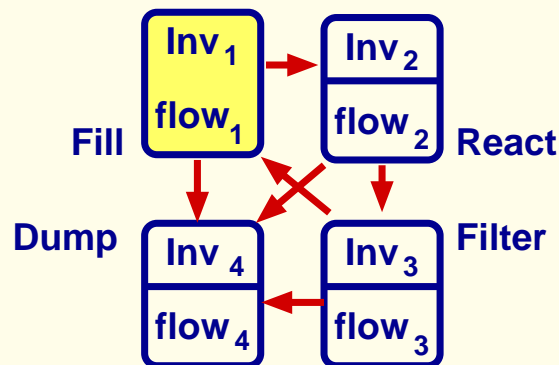
# Example



**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.

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

$$\text{flow}_1 \quad \dot{x}_1 \geq \text{dmin} \wedge \dot{x}_2 \geq \text{dmin} \wedge \dot{x}_3 = 0 \wedge -\delta_a \leq \dot{x}_1 - \dot{x}_2 \leq \delta_a$$



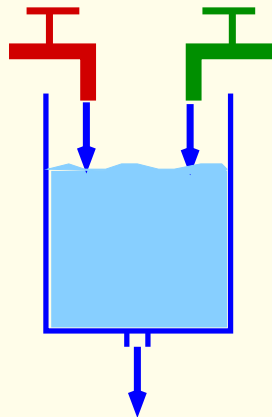
**Jumps: (1,2)**

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

$$e = (1, 2) \quad \text{guard}_{(1,2)}(x_1, x_2, x_3) = x_1 + x_2 + x_3 \geq L_f$$

$$\text{jump}_{(1,2)}(x_1, x_2, x_3, x'_1, x'_2, x'_3) = \bigwedge_{i=1}^3 x'_i = x_i$$

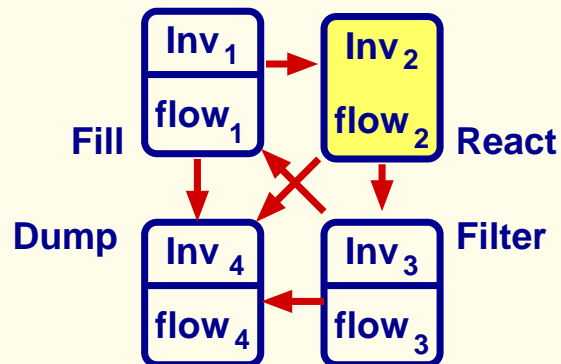
# Example



**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.

$$\text{Inv}_2 \quad L_f \leq x_1 + x_2 + x_3 \leq L_{\text{overflow}} \wedge \bigwedge_{i=1}^3 x_i \geq 0 \wedge -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \wedge 0 \leq x_3 \leq \text{max}$$

$$\text{flow}_2 \quad \dot{x}_1 \leq -\text{dmax} \wedge \dot{x}_2 \leq -\text{dmax} \wedge \dot{x}_3 \geq \text{dmin} \\ \wedge \dot{x}_1 = \dot{x}_2 \wedge \dot{x}_3 + \dot{x}_1 + \dot{x}_2 = 0$$

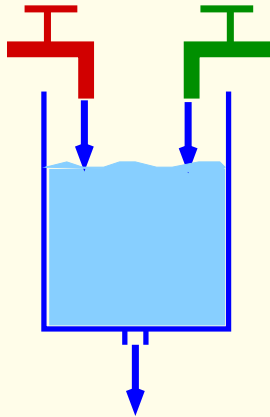


## Jumps:

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 minimal level min the system jumps into mode 3 (**Filter**).

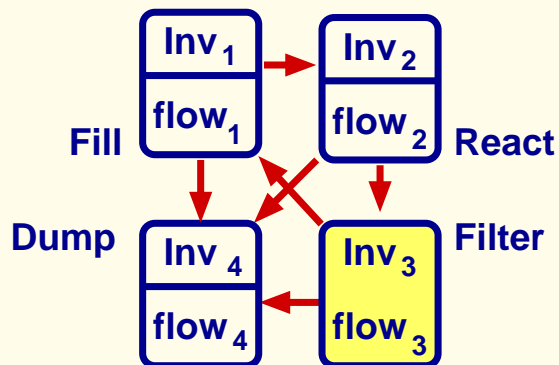
# Example



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

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

$$\text{flow}_3 \quad \dot{x}_1 = 0 \wedge \dot{x}_2 = 0 \quad \wedge \quad \dot{x}_3 \leq -d_{\max}$$

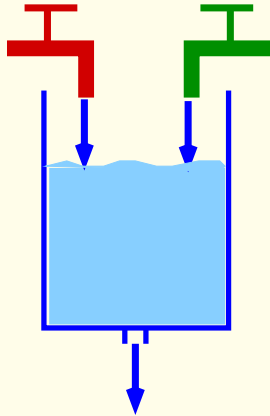


**Jumps:**

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

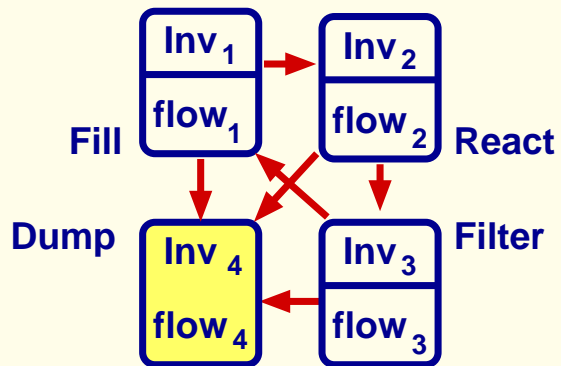
Otherwise, if the concentration of substance 3 is below some minimal level min the system jumps into mode 1 (**Fill**).

# Example



**Mode 4: Dump** The content of the tank is emptied.  
For simplicity we assume that this happens instantaneously:

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





# Simple verification problems

---

**Invariant checking:** Check whether  $\Psi$  is an invariant in a HA  $S$ , i.e.:

- (1)  $\text{Init}_q \models \Psi$  for all  $q \in Q$ ;
- (2)  $\Psi$  is invariant under jumps and flows:
  - (Flow)** For every flow in mode  $q$ , the continuous variables satisfy  $\Psi$  during and at the end of the flow.
  - (Jump)** For every jump according to a control switch  $e$ , if  $\Psi$  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:**  $\text{Inv}_q = \bigwedge_{j=1}^{m_q} (\sum_{i=1}^n a_{ij}^q x_i \leq a_j^q)$  can be expressed by:

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

# Simple verification problems

---

## Reductions of verification problems to linear arithmetic

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

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

# Simple verification problems

## Reductions of verification problems to linear arithmetic

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

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

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

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

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

# Simple verification problems

## Reductions of verification problems to linear arithmetic

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

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

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

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

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

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

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

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

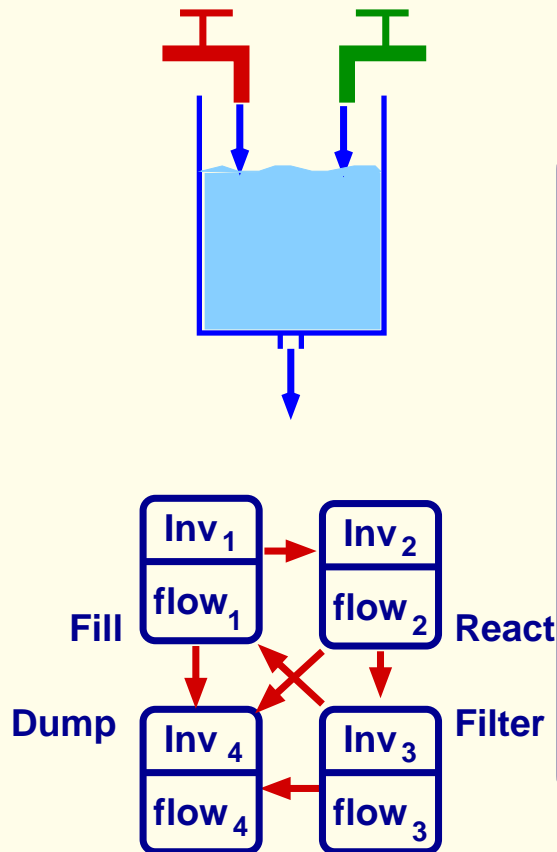
# Example

## 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):

$\Psi(0)$	$(x_1(0) + x_2(0) + x_3(0) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(0) - x_2(0) \leq \epsilon) \wedge$
$\neg\Psi(t)$	$\neg(x_1(t) + x_2(t) + x_3(t) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(t) - x_2(t) \leq \epsilon) \wedge$
$\text{Inv}_2(0)$	$L_f \leq x_1(0) + x_2(0) + x_3(0) \leq L_{\text{overflow}} \wedge x_3(0) \leq \text{max} \wedge$
$\text{Inv}_2(t)$	$L_f \leq x_1(t) + x_2(t) + x_3(t) \leq L_{\text{overflow}} \wedge x_3(t) \leq \text{max} \wedge$
$\text{flow}_2$	$x_1(t) - x_1(0) \leq -\text{dmax} \cdot t \wedge x_2(t) - x_2(0) \leq -\text{dmax} \cdot t \wedge$ $x_3(t) - x_3(0) \geq \text{dmin} \cdot t \wedge (x_1(t) - x_1(0)) - (x_2(t) - x_2(0)) = 0 \wedge$ $(x_1(t) - x_1(0)) + (x_2(t) - x_2(0)) + (x_3(t) - x_3(0)) = 0$



For fixed values for  $L_f, L_{\text{overflow}}$  – 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)



# Summary

---

- Basic notions in formal specification and verification

## Related topics

- Lecture "Decision Procedures for Verification"
- Seminar "Decision Procedures and Applications"

## More details on Specification, Model Checking, Verification:

This summer (probably end of August):

Summer school "Verification Technology, Systems & Applications"

**Forschungspraktika:** for instance SS 2017

**BSc/MSc Theses in the area**