Decision Procedures in Verification

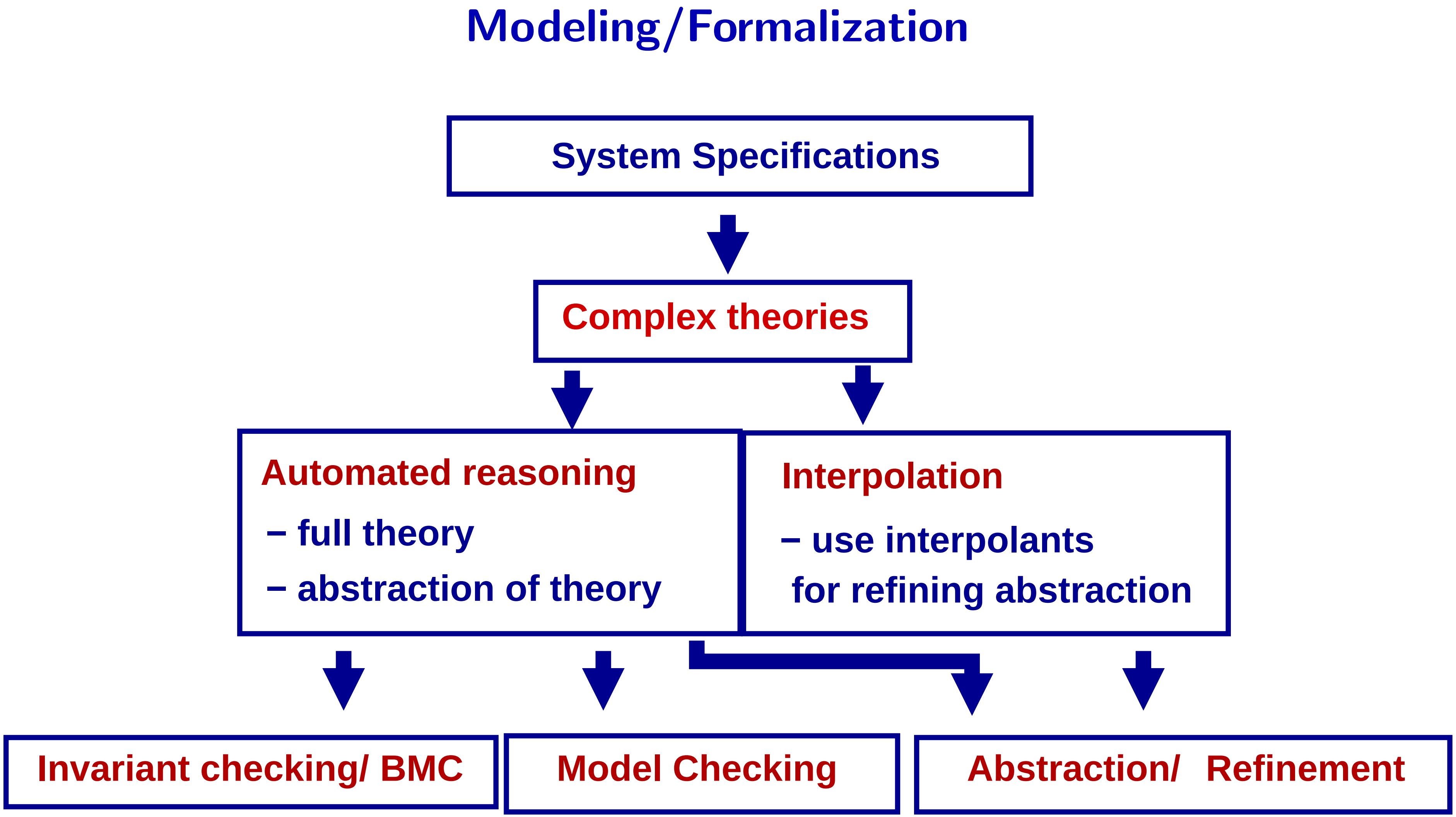
Applications

6.2.2014

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Verification



Examples: Verification

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

Given: 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 update)

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

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

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

Verification

Problems

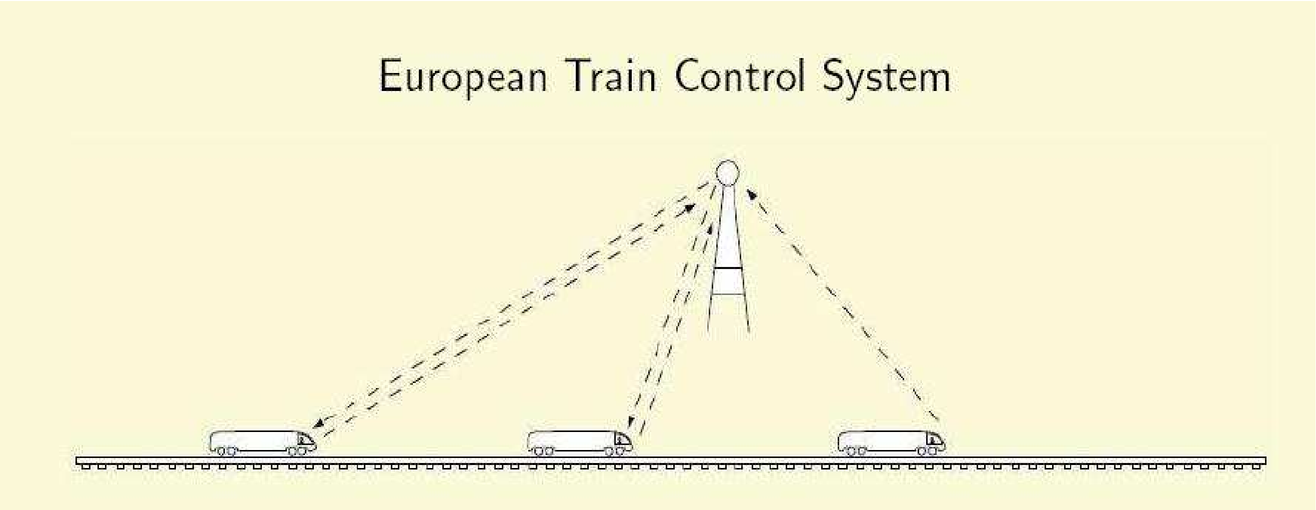
- Invariant checking, bounded model checking

Theories

- Theories of arrays
- Theories of pointer structures
- recursively defined functions
- sets
- ...

Example

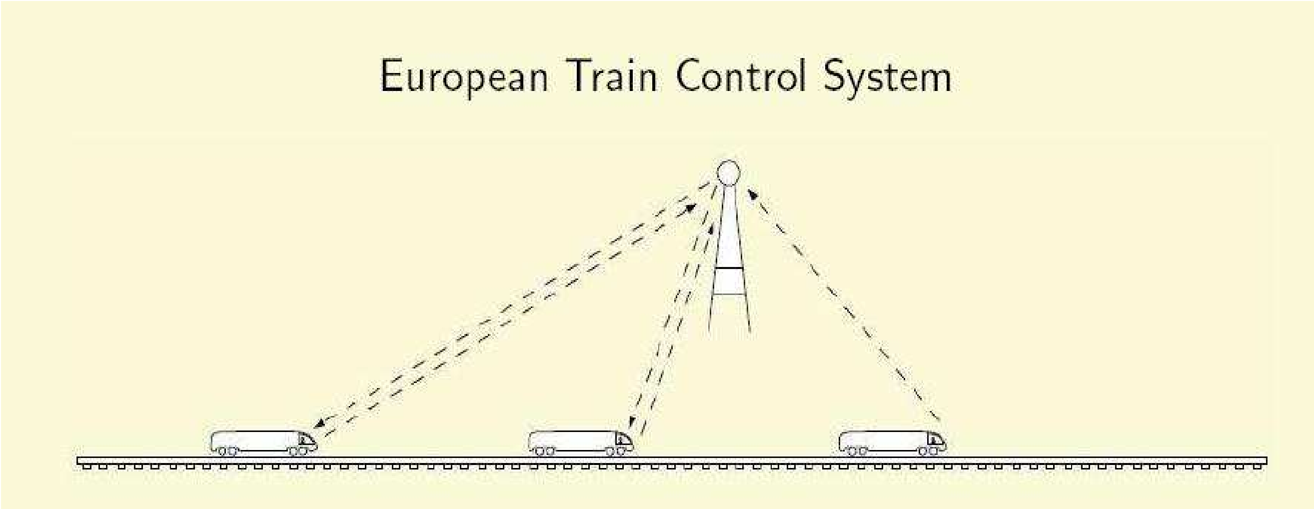
Simplified version of ETCS Case Study [Jacobs,VS'06, Faber,Jacobs,VS'07]



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:	$pos(i), pos'(i)$	$: \mathbb{Z} \to \mathbb{R}$

Example

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



Update(pos, pos$'$) :

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

...

Example

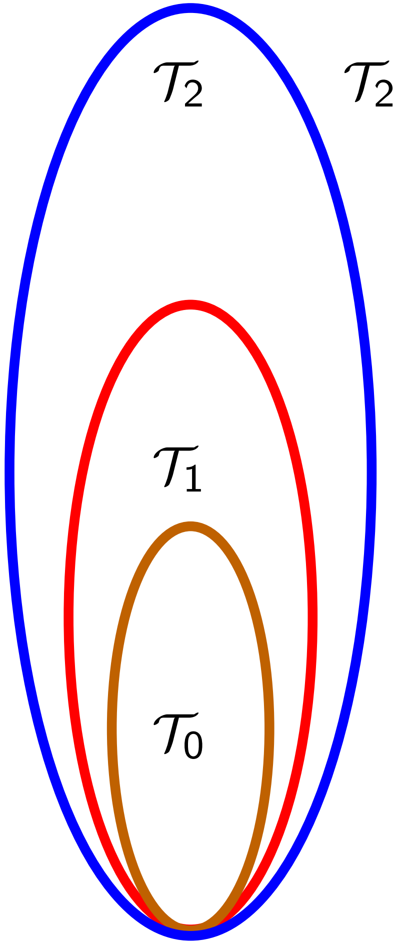
Safety property: No collisions $\quad \mathsf{Safe}(\mathsf{pos}): \quad \forall i, j (i < j \rightarrow \mathsf{pos}(i) > \mathsf{pos}(j))$

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

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

Our idea: Use chains of "instantiation" + reduction.

Example



$$\mathcal{T}_2 = \mathcal{T}_1 \cup \mathsf{Update}(\mathsf{pos}, \mathsf{pos}')$$

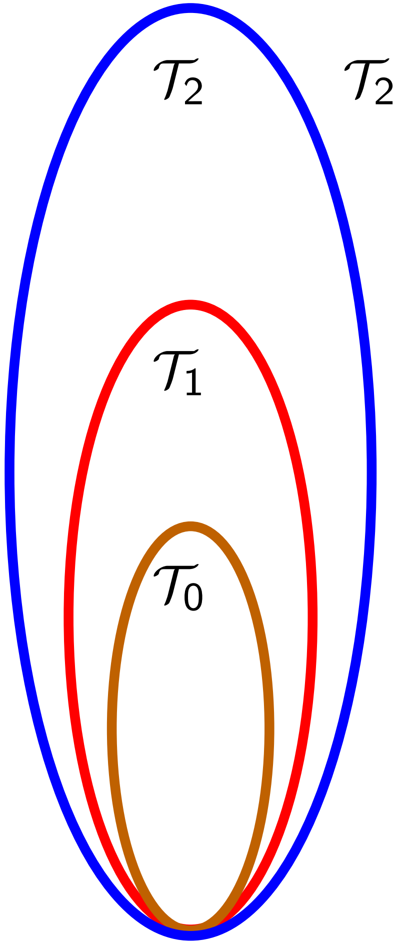
$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Safe}(\mathsf{pos})$$

$$\mathcal{T}_0 = \mathbb{R} \cup \mathbb{Z}$$

To show:

$$\mathcal{T}_2 \cup \underbrace{\neg \mathsf{Safe}(\mathsf{pos}')}_{G} \models \perp$$

Example



$\mathcal{T}_2 = \mathcal{T}_1 \cup \text{Update}(\text{pos}, \text{pos}')$

$\mathcal{T}_1 = \mathcal{T}_0 \cup \text{Safe}(\text{pos})$

$\mathcal{T}_0 = \mathbb{R} \cup \mathbb{Z}$

To show:

$$\mathcal{T}_2 \cup \underbrace{\neg \text{Safe}(\text{pos}')}_{G} \models \perp$$

$$\Downarrow$$

$$\mathcal{T}_1 \cup G'(\text{pos}) \models \perp$$

$$\Downarrow$$

$$\mathcal{T}_0 \cup G'' \models \perp$$

$$\Phi(c, \overline{c}_{\text{pos}'}, \overline{d}_{\text{pos}}, n, l_{\text{alarm}}, \text{min}, \text{max}, \Delta t) \models \perp$$

Method 1: SAT checking/ Counterexample generation

Method 2: Quantifier elimination

relationships between parameters which guarantee safety

More complex ETCS Case studies

[Faber, Jacobs, VS, 2007]

- Take into account also:
  - Emergency messages
  - Durations

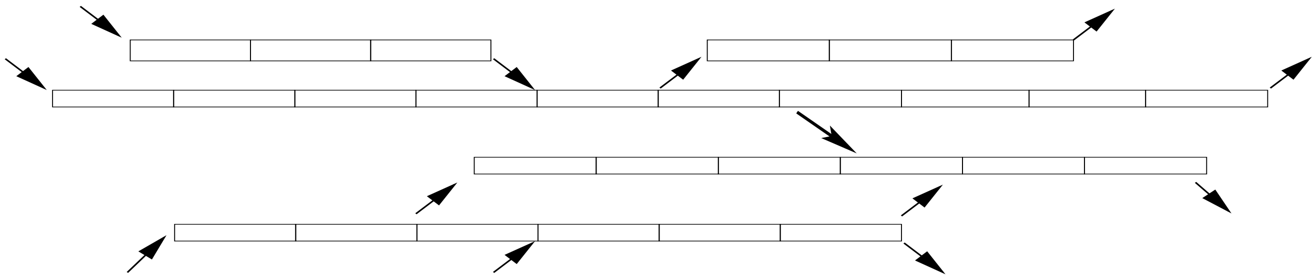
- Specification language: CSP-OZ-DC
  - Reduction to satisfiability in theories for which decision procedures exist

- **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]

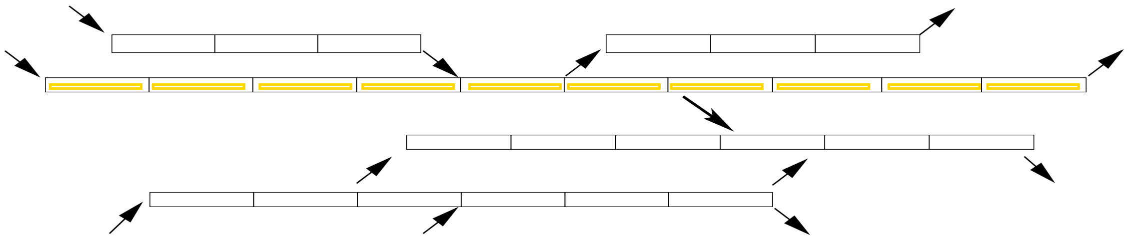


Assumptions:

- 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:

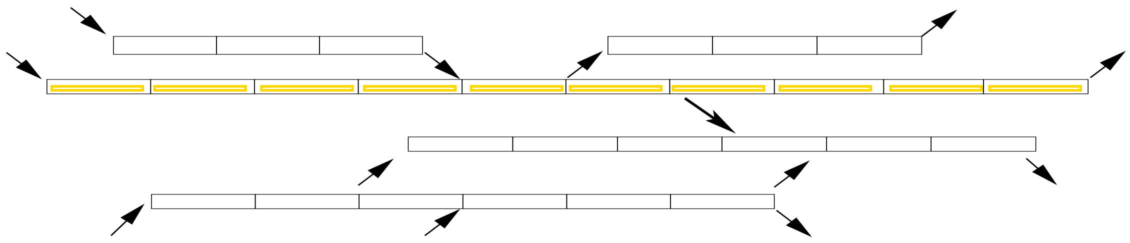
- No cycles
- in-degree (out-degree) of associated graph at most 2.

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:

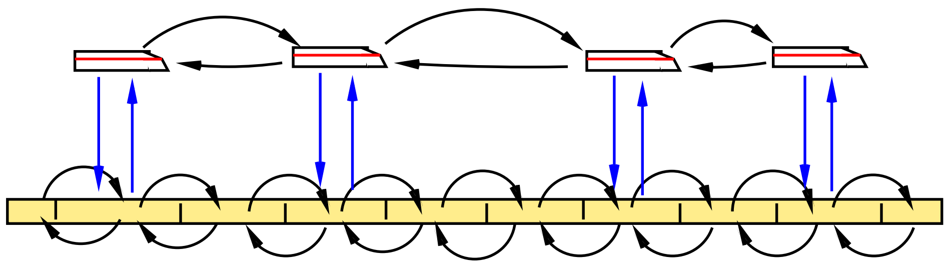
- No cycles
- in-degree (out-degree) of associated graph at most 2.

Data structures:

p_1: trains

- 2-sorted pointers

p_2: segments



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

Example: Controller for line track (RBC)

RBC

method $enter : [s1? : Segment;\ t0? : Train;\ t1? : Train;\ t2? : Train]$

method $leave : [ls? : Segment;\ lt? : Train]$

local_chan $alloc, req, updPos, updSpd$

$$
\begin{array}{lll}
\mathtt{main} & \stackrel{c}{=} & ((enter \to \mathtt{main}) \\
 & \Box & (leave \to \mathtt{main}) \\
 & \Box & (updSpd \to State1)) \\
State1 & \stackrel{c}{=} & ((enter \to State1) \\
 & \Box & (leave \to State1) \\
 & \Box & (req \to State2))
\end{array}
\qquad
\begin{array}{lll}
State2 & \stackrel{c}{=} & ((alloc \to State3) \\
 & \Box & (enter \to State2) \\
 & \Box & (leave \to State2)) \\
State3 & \stackrel{c}{=} & ((enter \to State3) \\
 & \Box & (leave \to State3) \\
 & \Box & (updPos \to \mathtt{main}))
\end{array}
$$

SegmentData

$train : Segment \to Train$	[Train on segment]
$req : Segment \to \mathbb{Z}$	[Requested by train]
$alloc : Segment \to \mathbb{Z}$	[Allocated by train]

TrainData

$segm : Train \to Segment$	[Train segment]
$next : Train \to Train$	[Next train]
$spd : Train \to \mathbb{R}$	[Speed]
$pos : Train \to \mathbb{R}$	[Current position]
$prev : Train \to Train$	[Prev. train]

$sd : SegmentData$
$td : TrainData$

$\forall t : Train \Gamma tid(t) > 0$
$\forall t1, t2 : Train \mid t1 \neq t2 \Gamma tid(t1) \neq tid(t2)$
$\forall s : Segment \Gamma prevs(nexts(s)) = s$
$\forall s : Segment \Gamma nexts(prevs(s)) = s$
$\forall s : Segment \Gamma sid(s) > 0$
$\forall s : Segment \Gamma sid(nexts(s)) > sid(s)$
$\forall s1, s2 : Segment \mid s1 \neq s2 \Gamma sid(s1) \neq sid(s2)$
$\forall s : Segment \mid s \neq snil \Gamma length(s) > d + gmax \cdot \Delta t$
$\forall s : Segment \mid s \neq snil \Gamma 0 < lmax(s) \wedge lmax(s) \leq gmax$
$\forall s : Segment \Gamma lmax(s) \geq lmax(prevs(s)) - decmax \cdot \Delta t$
$\forall s1, s2 : Segment \Gamma tid(incoming(s1)) \neq tid(train(s2))$

Init

$\forall t : Train \Gamma train(segm(t)) = t$
$\forall t : Train \Gamma next(prev(t)) = t$
$\forall t : Train \Gamma prev(next(t)) = t$
$\forall t : Train \Gamma 0 \leq pos(t) \leq length(segm(t))$
$\forall t : Train \Gamma 0 \leq spd(t) \leq lmax(segm(t))$
$\forall t : Train \Gamma alloc(segm(t)) = tid(t)$
$\forall t : Train \Gamma alloc(nexts(segm(t))) = tid(t)$
$\quad \vee\ length(segm(t)) - bd(spd(t)) > pos(t)$
$\forall s : Segment \Gamma segm(train(s)) = s$

effect_updSpd

$\Delta(spd)$

$\forall t : Train \mid pos(t) < length(segm(t)) - d \wedge spd(t) - decmax \cdot \Delta t > 0$
$\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq lmax(segm(t)))$
$\forall t : Train \mid pos(t) \geq length(segm(t)) - d \wedge alloc(nexts(segm(t))) = tid(t)$
$\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq \min\{lmax(segm(t)), lmax(nexts(segm(t)))\}$
$\forall t : Train \mid pos(t) \geq length(segm(t)) - d \wedge \neg\ alloc(nexts(segm(t))) = tid(t)$
$\quad \Gamma spd'(t) = \max\{0, spd(t) - decmax \cdot \Delta t\}$

$\vdots$

CSP

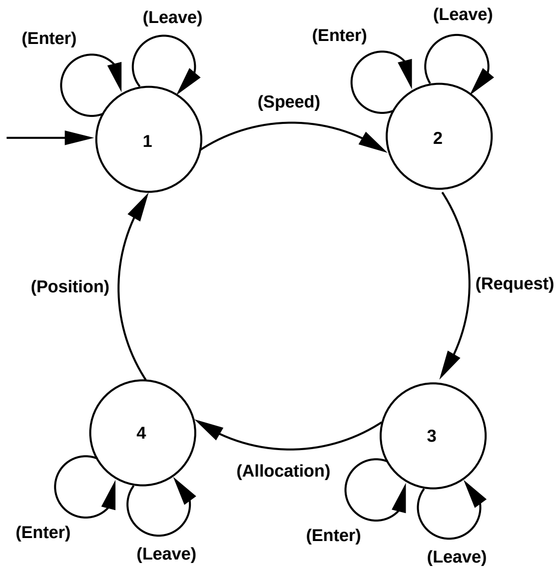
OZ

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)

CSP part: specifies the processes and their interdependency.

The RBC system passes repeatedly through four phases, modeled by events with corresponding COD schemata:

CSP:

$\texttt{method}\ \mathit{enter} : [s1? : \mathit{Segment};\ t0? : \mathit{Train};\ t1? : \mathit{Train};\ t2? : \mathit{Train}]$

$\texttt{method}\ \mathit{leave} : [ls? : \mathit{Segment};\ lt? : \mathit{Train}]$

$\texttt{local\_chan}\ \mathit{alloc}, \mathit{req}, \mathit{updPos}, \mathit{updSpd}$

$$\begin{array}{llll}
\texttt{main} \stackrel{c}{=} ((\mathit{updSpd} \to \mathit{State}1) & \mathit{State}1 \stackrel{c}{=} ((\mathit{req} \to \mathit{State}2) & \mathit{State}2 \stackrel{c}{=} ((\mathit{alloc} \to \mathit{State}3) & \mathit{State}3 \stackrel{c}{=} ((\mathit{updPos} \to \texttt{main}) \\
\quad \Box (\mathit{leave} \to \texttt{main}) & \quad \Box (\mathit{leave} \to \mathit{State}1) & \quad \Box (\mathit{leave} \to \mathit{State}2) & \quad \Box (\mathit{leave} \to \mathit{State}3) \\
\quad \Box (\mathit{enter} \to \texttt{main})) & \quad \Box (\mathit{enter} \to \mathit{State}1)) & \quad \Box (\mathit{enter} \to \mathit{State}2)) & \quad \Box (\mathit{enter} \to \mathit{State}3))
\end{array}$$

Example: Controller for line track (RBC)

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

Example: Controller for line track (RBC)

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

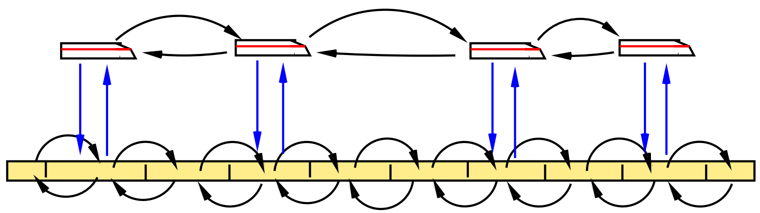
- 1. Data classes declare function symbols that can change their values during runs of the system

Data structures:

- 2-sorted pointers

train: trains

segm: segments



SegmentData

$train : Segment \to Train$	
	[Train on segment]
$req : Segment \to \mathbb{Z}$	[Requested by train]
$alloc : Segment \to \mathbb{Z}$	
	[Allocated by train]

TrainData

$segm : Train \to Segment$	
	[Train segment]
$next : Train \to Train$	[Next train]
$spd : Train \to \mathbb{R}$	[Speed]
$pos : Train \to \mathbb{R}$	[Current position]
$prev : Train \to Train$	[Prev. train]

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the `Init` schema, 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` schema, 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

$$
\begin{array}{l}
\texttt{effect\_}updSpd \\
\Delta(spd) \\
\hline
\forall\, t : Train \mid pos(t) < length(segm(t)) - d \wedge spd(t) - decmax \cdot \Delta t > 0 \\
\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq lmax(segm(t)) \\
\forall\, t : Train \mid pos(t) \geq length(segm(t)) - d \wedge alloc(nexts(segm(t))) = tid(t) \\
\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq \min\{lmax(segm(t)), lmax(nexts(segm(t)))\} \\
\forall\, t : Train \mid pos(t) \geq length(segm(t)) - d \wedge \neg\, alloc(nexts(segm(t))) = tid(t) \\
\quad \Gamma spd'(t) = \max\{0, spd(t) - decmax \cdot \Delta t\}
\end{array}
$$

Modular Verification

COD specification	$\mapsto$ Σ_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)

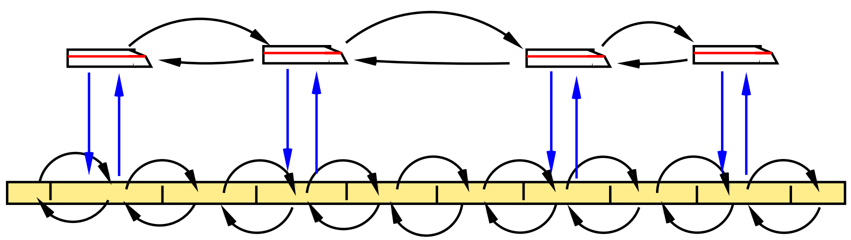
  (2) $\models_{\mathcal{T}_S} \mathsf{Safe}(\overline{x}) \wedge \mathsf{Update}(\overline{x}, \overline{x}') \rightarrow \mathsf{Safe}(\overline{x}')$ (Safe holds before $\Rightarrow$ holds after update)

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

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

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

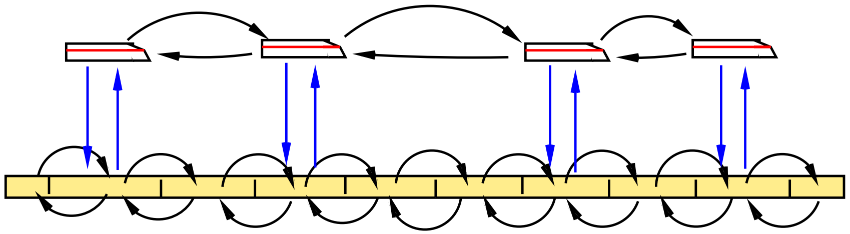
Trains on a linear track



Example 1: Speed Update

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

Trains on a linear track



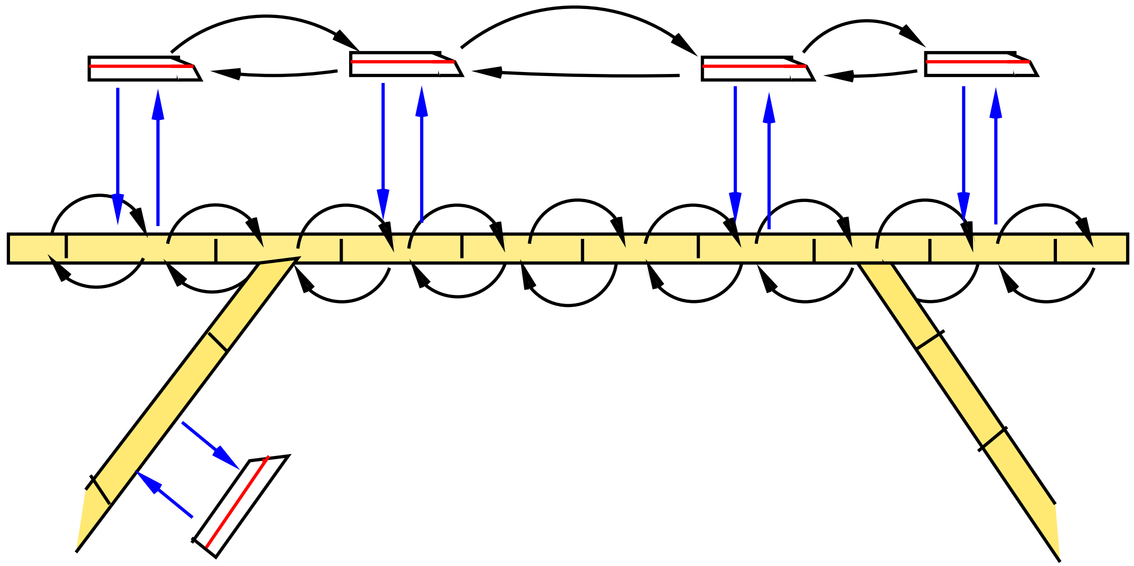
Example 1: Speed Update

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

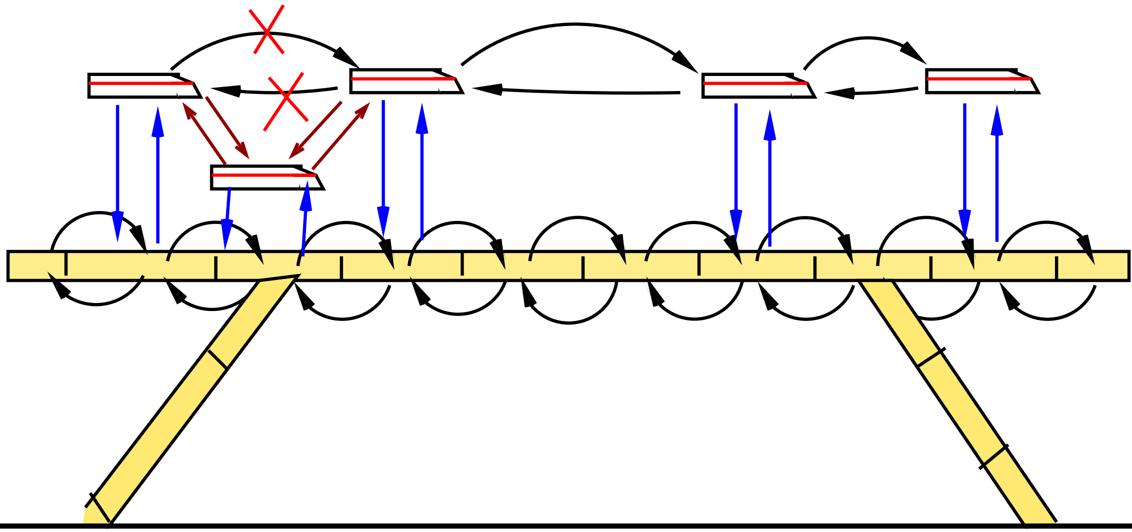
Proof task:

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

Incoming and outgoing trains



Incoming and outgoing trains



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

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

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

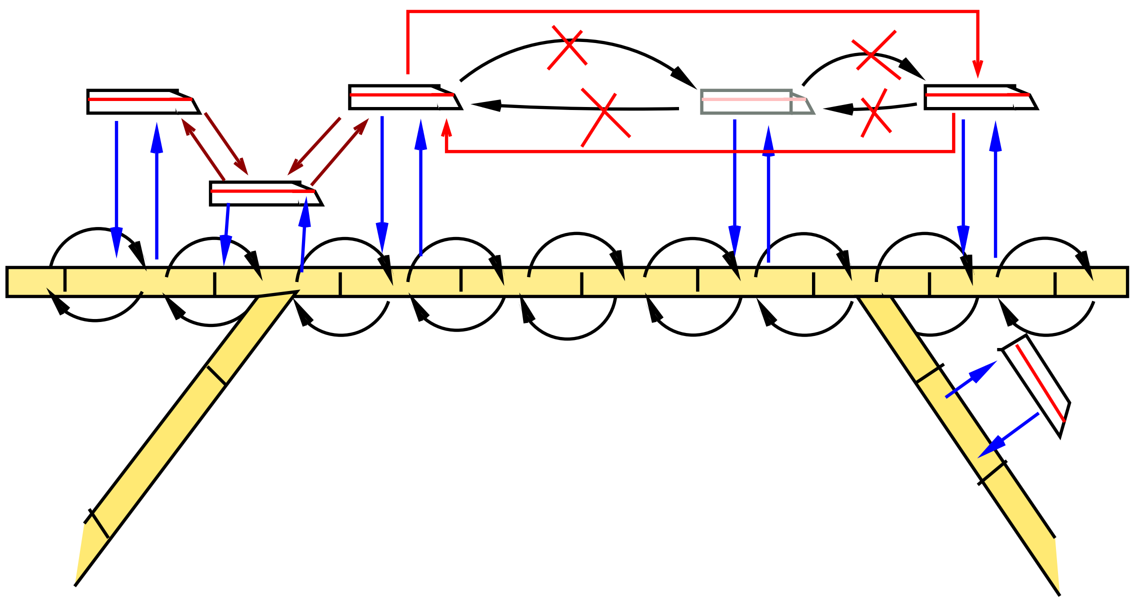
$t{\neq}t_1$, $\text{ids}(\text{segm}(t)){<}\text{ids}(s_1)$, $\text{alloc}(s_1){=}\text{tid}(t_1)$, $\text{next}_t(t){\neq}\text{null}_t$, $\text{ids}(\text{segm}(\text{next}_t(t))){\leq}\text{ids}(s_1)$

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

...

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

Incoming and outgoing trains



Safety property

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

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

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:

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

and that the invariants are preserved under all transitions of the system,

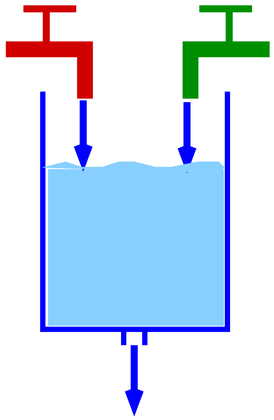
$$(\text{Inv}(i)) \wedge (\text{Update}) \models (\text{Inv}'(j))$$

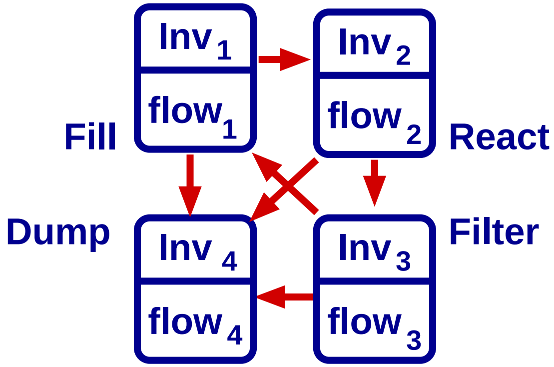
whenever (Update) is a transition from location i to j .

Other Applications

Verification of “Hybrid automata”

Example





Chemical plant

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

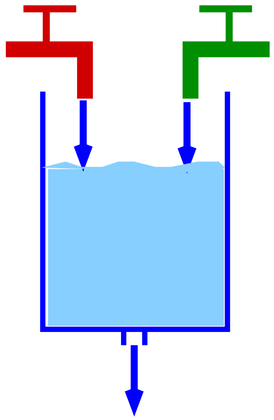
Check:

- 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

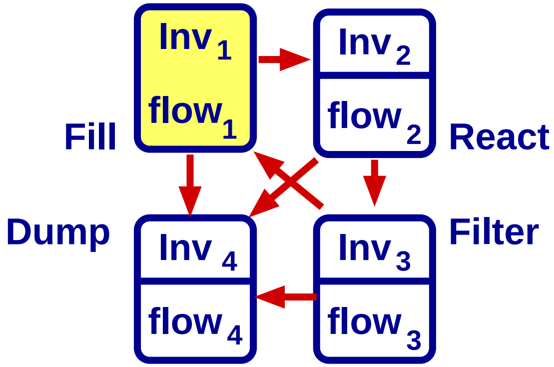
Example



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	$x_1 + x_2 + x_3 \leq L_f \;\wedge\; \bigwedge_{i=1}^{3} x_i \geq 0 \;\wedge$ $-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \;\wedge\; 0 \leq x_3 \leq \text{min}$
flow_1	$\dot{x}_1 \geq \text{dmin} \wedge \dot{x}_2 \geq \text{dmin} \wedge \dot{x}_3 = 0 \wedge -\delta_a \leq \dot{x}_1 - \dot{x}_2 \leq \delta_a$

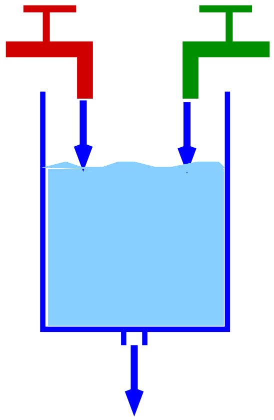


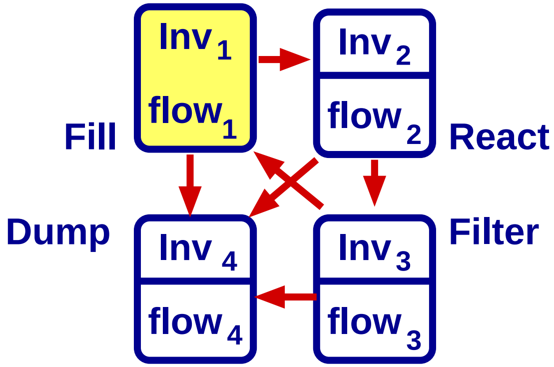
Jumps: (1,4)

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

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

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.

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

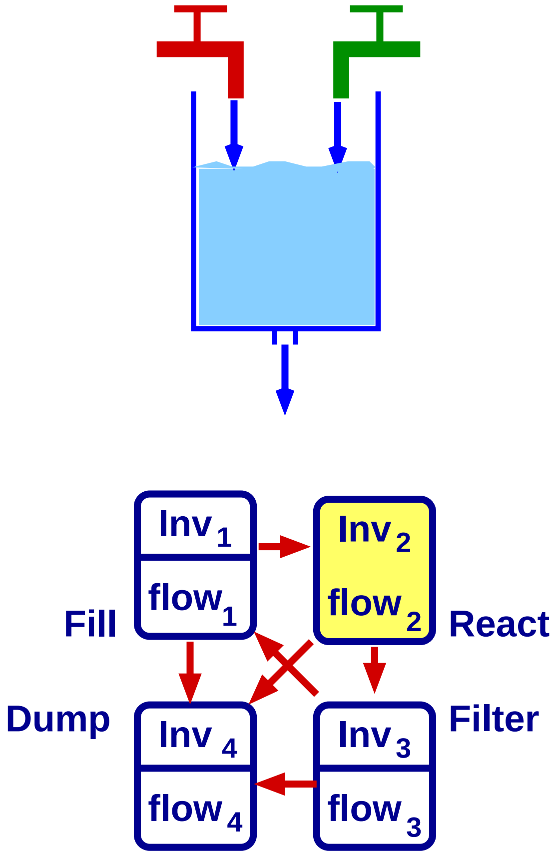
$$\text{flow}_1 \quad \dot{x}_1 \geq \text{dmin} \wedge \dot{x}_2 \geq \text{dmin} \wedge \dot{x}_3 = 0 \wedge -\delta_a \leq \dot{x}_1 - \dot{x}_2 \leq \delta_a$$

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.

Inv_2 $\quad L_f \leq x_1 + x_2 + x_3 \leq L_{\mathrm{overflow}} \;\wedge\; \bigwedge_{i=1}^{3} x_i \geq 0 \;\wedge$
$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \;\wedge\; 0 \leq x_3 \leq \mathrm{max}$

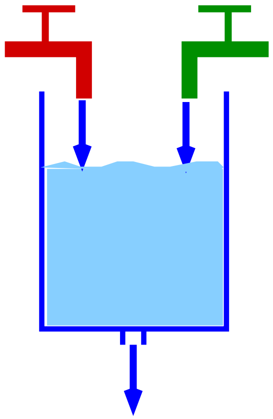
flow_2 $\quad \dot{x}_1 \leq -\mathrm{dmax} \wedge \dot{x}_2 \leq -\mathrm{dmax} \wedge \dot{x}_3 \geq \mathrm{dmin}$
$\wedge \dot{x}_1 = \dot{x}_2 \wedge \dot{x}_3 + \dot{x}_1 + \dot{x}_2 = 0$

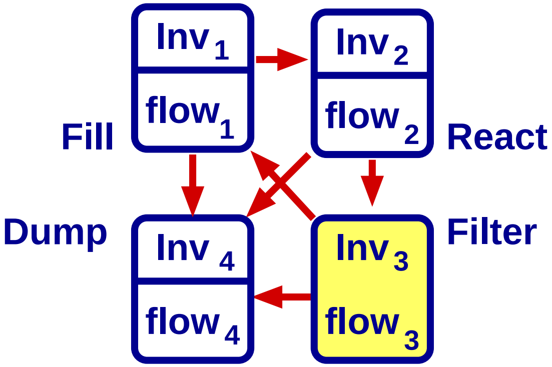
Jumps:

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.

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

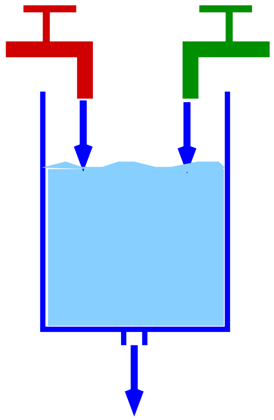
$$\mathrm{flow}_3 \qquad \dot{x}_1 = 0 \wedge \dot{x}_2 = 0 \;\wedge\; \dot{x}_3 \leq -\mathrm{dmax}$$

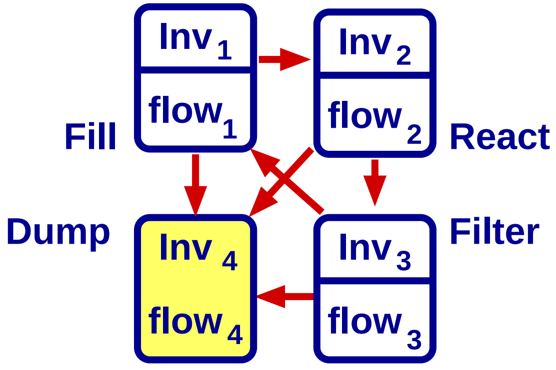
Jumps:

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:

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

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:

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

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

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

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

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

Invariant checking

Theorem. The following are equivalent for any LHA:

(1) Ψ (a convex predicate) is an invariant of the LHA;

(2) For all $q \in Q, e = (q, q') \in E$, the following are unsatisfiable:

$$F_{\text{Flow}}(q) \qquad \Psi(\overline{x}(t_0)) \wedge \text{Flow}_q(t_0, t) \wedge \neg\Psi(\overline{x}(t)) \wedge t \geq t_0$$

$$F_{\text{jump}}(e) \qquad \Psi(\overline{x}(t)) \wedge \text{Jump}_e(\overline{x}(t), \overline{x}'(0)) \wedge \text{Inv}_{q'}(\overline{x}'(0)) \wedge \neg\Psi(\overline{x}'(0))$$

(3) For all $q \in Q, e = (q, q') \in E$, the following are unsatisfiable:

$$F_{\text{flow}}(q) \qquad \Psi(\overline{x}(t_0)) \wedge \text{Inv}_q(\overline{x}(t_0)) \wedge \underline{\text{flow}}_q(t_0, t) \wedge \text{Inv}_q(\overline{x}(t)) \wedge \neg\Psi(\overline{x}(t)) \wedge t \geq t_0$$

$$F_{\text{jump}}(e) \qquad \Psi(\overline{x}(t)) \wedge \text{Jump}_e(\overline{x}(t), \overline{x}'(0)) \wedge \text{Inv}_{q'}(\overline{x}'(0)) \wedge \neg\Psi(\overline{x}'(0))$$

- $\text{Flow}_q(t_0, t)$: $\quad \forall\, t' (t_0 \leq t' \leq t \rightarrow \text{Inv}_q(\overline{x}(t'))) \quad \wedge \quad \forall\, t', t'' (t_0 \leq t' \leq t'' \leq t \rightarrow \underline{\text{flow}}_q(t', t''))$.
- $\underline{\text{flow}}_q(t_0, t) = \bigwedge_{j=1}^{n_q} (\sum_{i=1}^{n} c_{ij}^q (x_i(t) - x_i(t_0)) \leq c_j^q (t - t_0))$.

Invariant checking

Theorem. The following are equivalent for any LHA:

(1) Ψ (a convex predicate) is an invariant of the LHA;

(

Invariant checking: Reduction to checking the satisfiability of conjunctions of linear inequalities $\mapsto$ can be checked in PTIME [Khachian]

Parametric systems: Use QE to generate constraints on parameters which guarantee that Ψ invariant $\mapsto$ can be done in EXPTIME in general; if constraints in $UTVPI^{\neq}$: PTIME [Koubarakis, Skiadoupoulos]

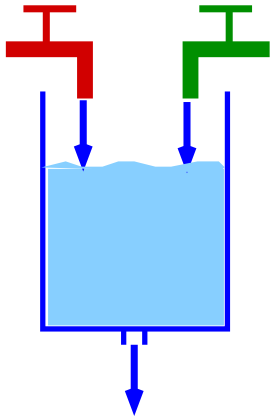
(3) For all $q \in Q$, $e = (q, q') \in E$, the following are unsatisfiable:

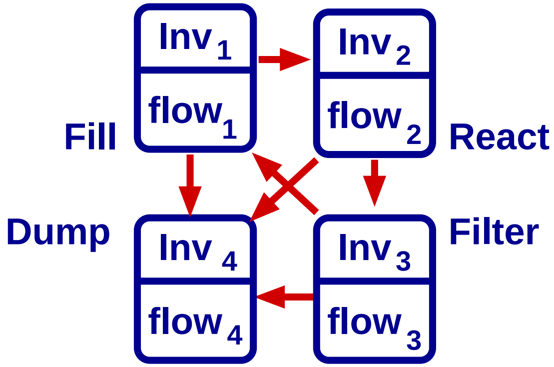
$$F_{\text{flow}}(q) \qquad \Psi(\overline{x}(t_0)) \wedge \mathsf{Inv}_q(\overline{x}(t_0)) \wedge \underline{\mathsf{flow}}_q(t_0, t) \wedge \mathsf{Inv}_q(\overline{x}(t)) \wedge \neg\Psi(\overline{x}(t)) \wedge t \geq t_0$$

$$F_{\text{jump}}(e) \qquad \Psi(\overline{x}(t)) \wedge \mathsf{Jump}_e(\overline{x}(t), \overline{x}'(0)) \wedge \mathsf{Inv}_{q'}(\overline{x}'(0)) \wedge \neg\Psi(\overline{x}'(0))$$

$$\underline{\mathsf{flow}}_q(t_0, t_1) = \bigwedge_{j=1}^{n_q} \left(\sum_{i=1}^{n} c_{ij}^q (x_i(t_1) - x_i(t_0)) \leq c_j^q (t_1 - t_0)\right).$$

Example





Invariant:

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

We assume that $L_f < L_{\text{overflow}}$ and $\epsilon_a < \epsilon$.

ϕ_{safe} is an invariant iff

- for every mode $q \in \{1, 2, 3, 4\}$ $F_{\text{flow}}(q)$ unsat.:

  $\phi_{\text{safe}}(\overline{x}(0)) \wedge \text{Inv}_q(\overline{x}(0)) \wedge \underline{\text{flow}}_q(\overline{x}, t) \wedge \text{Inv}_q(\overline{x}(t)) \wedge \neg\phi_{\text{safe}}(\overline{x}(t))$

- $F_{\text{Jump}}(e)$ is unsatisfiable for all $e \in E$.

Safety property

Need additional invariants.

- generate by hand [Faber, Ihlemann, Jacobs, VS, ongoing]

  use the capabilities of H-PILoT of generating counterexamples

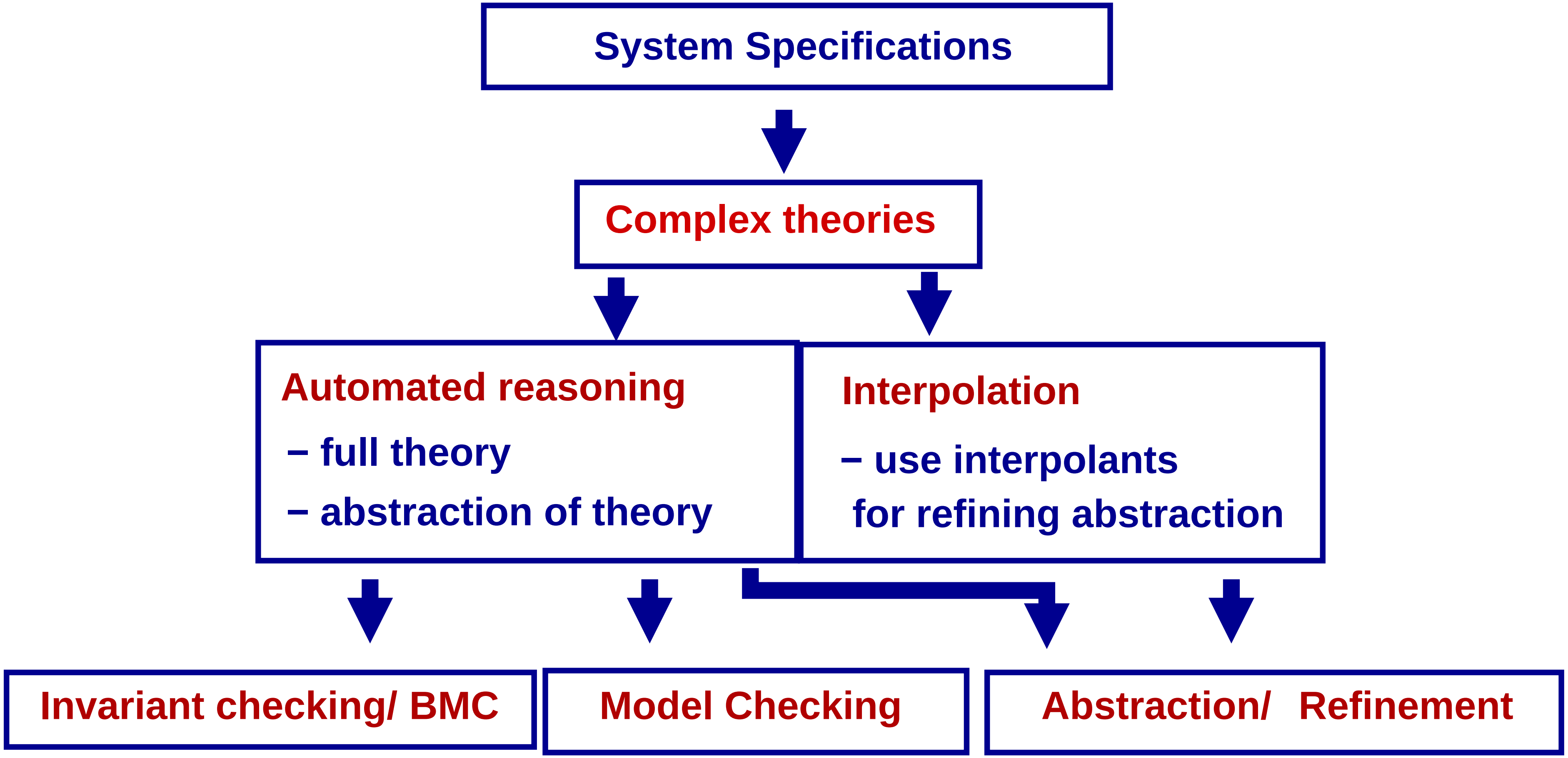
- generate automatically [VS, work in progress]

Ground satisfiability problems for pointer data structures

the decision procedures presented before can be used without problems

Verification

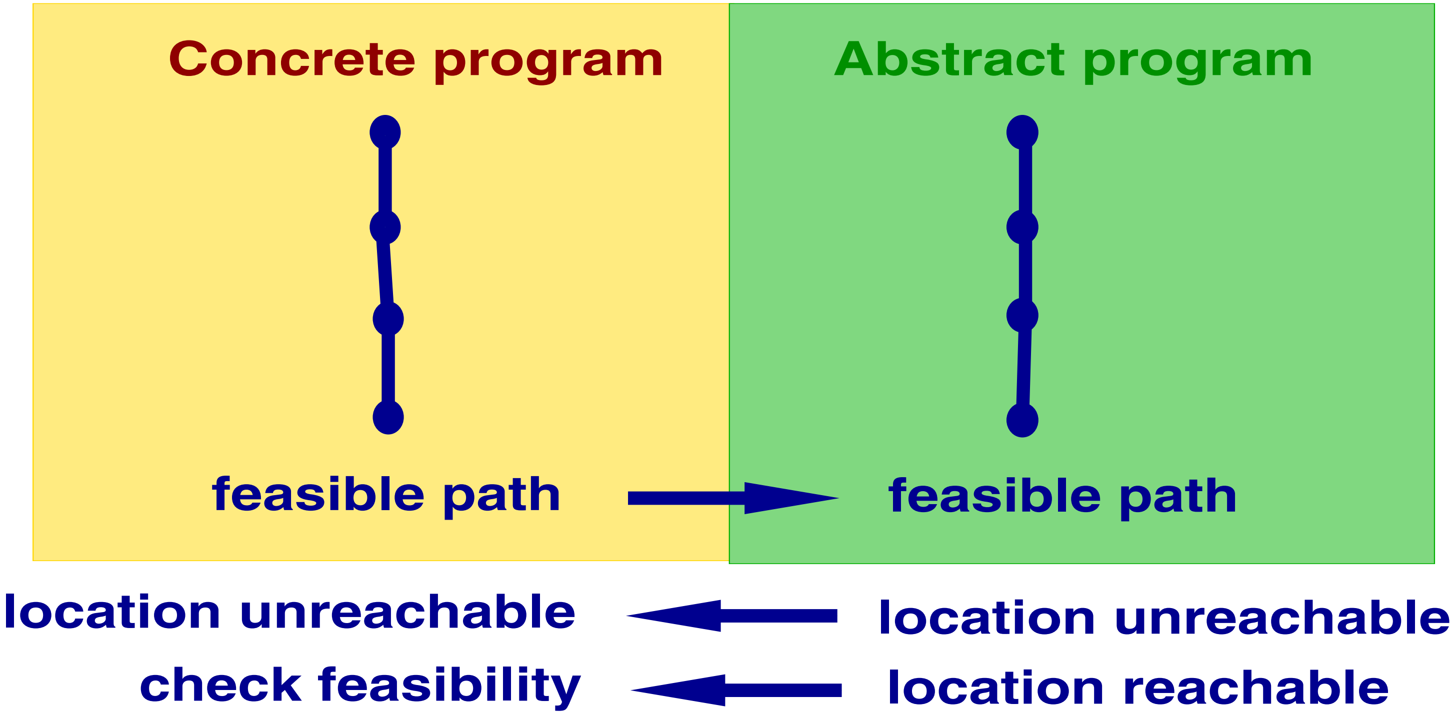
Modeling/Formalization



Other interesting topics

- Generate invariants
- Verification by abstraction/refinement

Abstraction-based Verification



$\Downarrow$

conjunction of constraints: $\phi(1) \wedge Tr(1,2) \wedge \cdots \wedge Tr(n-1,n) \wedge \neg \mathsf{safe}(n)$

- satisfiable: feasible path
- unsatisfiable: refine abstract program s.t. the path is not feasible

[McMillan 2003-2006] use 'local causes of inconsistency'

$\mapsto$ compute interpolants

Summary

- Decision procedures for various theories/theory combinations

  Implemented in most of the existing SMT provers:

  Z3: `http://z3.codeplex.com/`

  CVC4: `http://cvc4.cs.nyu.edu/web/`

  Yices: `http://yices.csl.sri.com/`

- Ideas about how to use them for verification

More details on Specification, Model Checking, Verification:

Next semester: Formal Specification and Verification

Decision procedures for other classes of theories/Applications"

Next semester: Seminar "Decision Procedures and Applications"

Forschungspraktikum

BSc/MSc Theses in the area