Using decision procedures for rich data structures for the verification of real-time systems

Viorica Sofronie-Stokkermans

Joint work with Johannes Faber, Carsten Ihlemann, Swen Jacobs
and with Werner Damm, Matthias Horbach

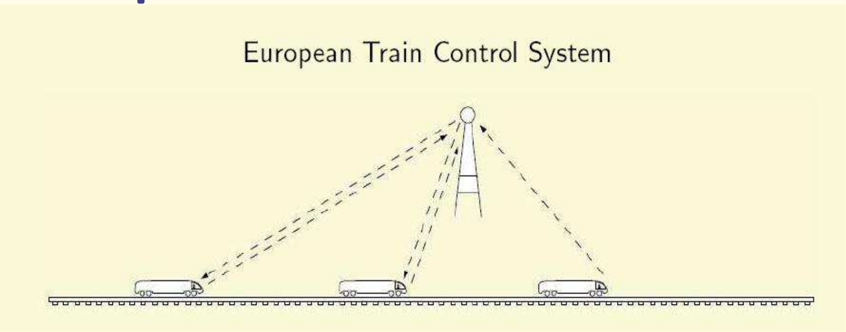
AVACS Concluding Colloquium, Oldenburg, 29 September 2015.

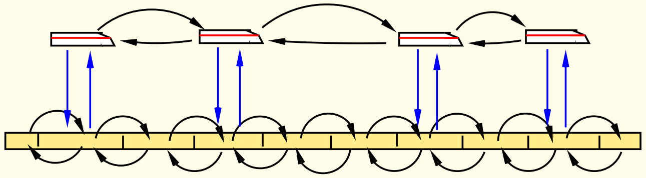
Problem statement

We consider parametric real time   (infinite state) systems

– parametric data, parametric change, parametric topology of the system

Examples:





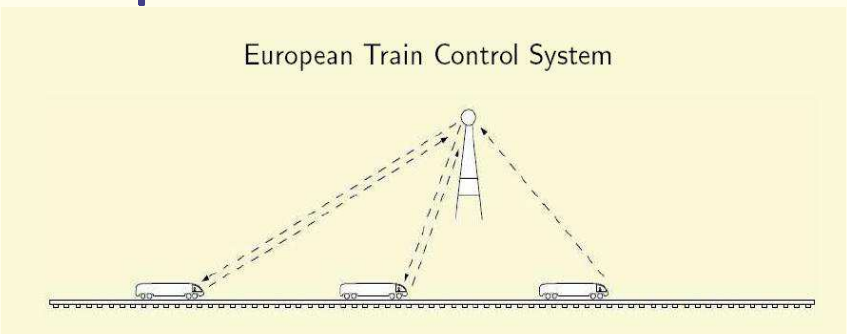
n (number of trains); $l_{alarm} > 0$; $0 < v_{min} < v_{max}$; ...

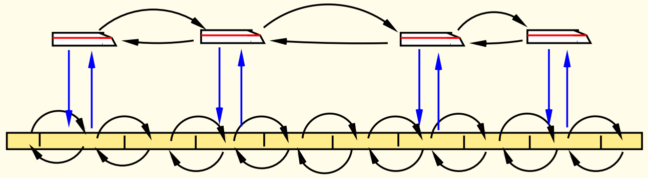
Problem statement

We consider parametric real time and hybrid (infinite state) systems

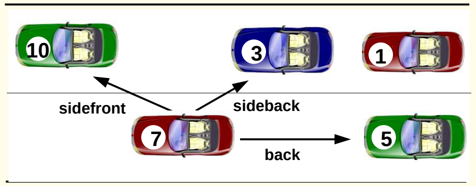
– parametric data, parametric change, parametric topology of the system

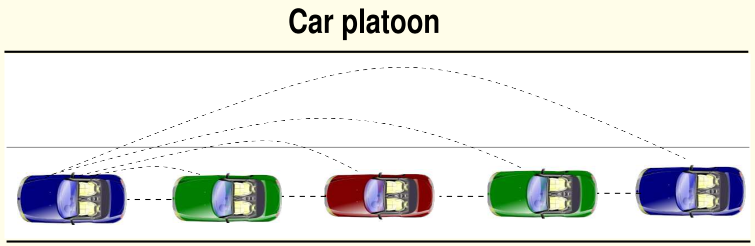
Examples:





n (number of trains); $l_{\text{alarm}} > 0$; $0 < v_{\text{min}} < v_{\text{max}}$; ...





Main results

Our work in AVACS (R1)

- **Specification of systems with a complex topology**
  data structures (arrays, pointer structures)
- **Deductive verification:** Invariant checking, BMC, Constraints on parameters
  (using decision procedures for rich data structures, quantifier elimination)
  [Jacobs,VS: PDPAR'06, ENTCS'07], [Faber,Jacobs,VS: IFM'07],
  [Faber,Ihlemann,Jacobs,VS: IFM'10], [VS: IJCAR'10], [VS: CADE'13]

$\mapsto$ **Efficient decision procedures for data structures**

local theory extensions [VS: CADE'05, FroCoS'07]
- ordered structures [Ihlemann,VS: ISMVL'07]
- theories of arrays & pointers [Ihlemann,Jacobs,VS: TACAS'08]
- theories from mathematical analysis [VS: KI'08]
- combinations of local theory extensions [Ihlemann,VS: IJCAR'10], [VS: PL'13]

$\mapsto$ **Interpolation in local theory extensions** $\mapsto$ **CEGAR**

[VS: IJCAR'06, LMCS'08], [Rybalchenko,VS: VMCAI'07, JSC'10], [VS: PL'13]

State of the art/Main results

We consider parametric real time and hybrid (infinite state) systems
– parametric data, parametric change, parametric topology

Previous work often only few aspects of parametricity studied together
approximations/abstraction

Before [Jacobs,VS'06, '07], [Faber, Jacobs, VS'07], [Faber, Ihlemann, Jacobs, VS'10]:

- only parametricity in the data domain: [Platzer, Quesel'09]
- parametric number of components:
  [Abdulla et al.'98] timed automata; [Arons et al.'01] finite-state systems

Before [VS: CADE'13], [Damm,Horbach,VS: FroCoS'15]:

- modularity and small model property results for restricted classes of systems
  [Kaiser, Kroening et al.'10], [Johnson,Mitra'12], [Abdulla et al'13]

State of the art/Main results

We consider parametric real time and hybrid (infinite state) systems
– parametric data, parametric change, parametric topology

Previous work often only few aspects of parametricity studied together
approximations/abstraction

Before [Jacobs,VS'06, '07], [Faber, Jacobs, VS'07], [Faber, Ihlemann, Jacobs, VS'10]:

- only parametricity in the data domain: [Platzer, Quesel'09]
- parametric number of components:
  [Abdulla et al.'98] timed automata; [Arons et al.'01] finite-state systems

Before [VS: CADE'13], [Damm,Horbach,VS: FroCoS'15]:

- modularity and small model property results for restricted classes of systems
  [Kaiser, Kroening et al.'10], [Johnson,Mitra'12], [Abdulla et al'13]

Our main goal: Reduce complexity by exploiting modularity at various levels: specification / verification / structurally

State of the art/Main results

We consider parametric real time and hybrid (infinite state) systems
– parametric data, parametric change, parametric topology

Previous work often only few aspects of parametricity studied together
approximations/abstraction

Before [Jacobs,VS'06, '07], [Faber, Jacobs, VS'07], [Faber, Ihlemann, Jacobs, VS'10]:

- only parametricity in the data domain: [Platzer, Quesel'09]
- parametric number of components:
  [Abdulla et al.'98] timed automata; [Arons et al.'01] finite-state systems

Before [VS: CADE'13], [Damm,Horbach,VS: FroCoS'15]:

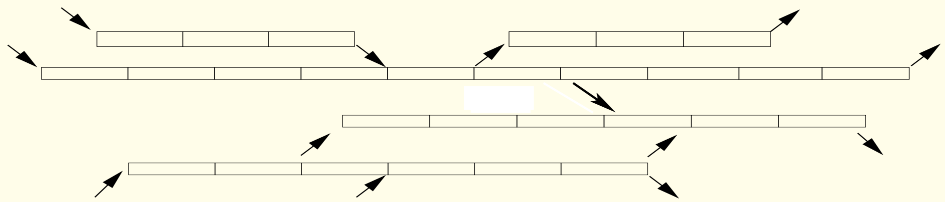
- modularity and small model property results for restricted classes of systems
  [Kaiser, Kroening et al.'10], [Johnson,Mitra'12], [Abdulla et al'13]

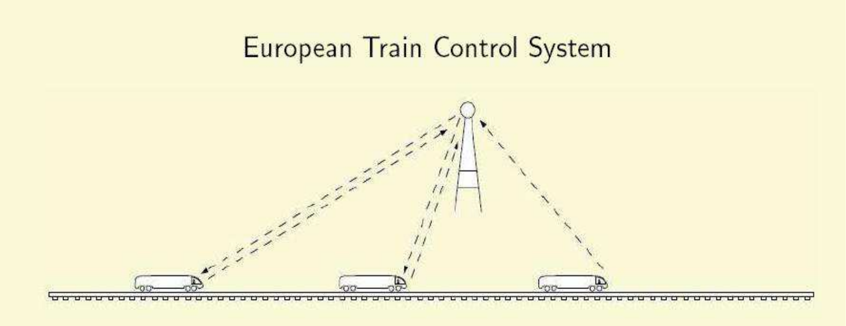
Our main goal: Reduce complexity by exploiting modularity at various levels:
specification / verification / structurally

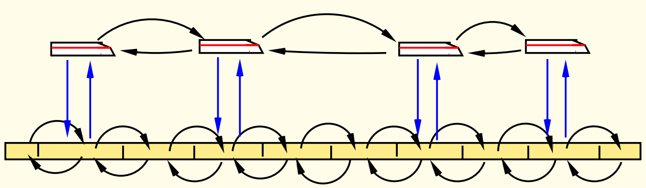
Example 1: Verification of systems of trains

[Faber,Ihlemann,Jacobs,VS 2010]

J. Faber, C. Ihlemann, S. Jacobs, V. Sofronie-Stokkermans:
Automatic Verification of Parametric Specifications with Complex Topologies. Proc. IFM 2010, LNCS 6396, 2010, pp 152-167







Main goal: exploit modularity at various levels

1. Specification

- Use the modular language COD, which allows us to separately specify
  - processes (as Communicating Sequential Processes, CSP),
  - data (using Object-Z, OZ), and
  - time (using the Duration Calculus, DC).

Main goal: exploit modularity at various levels

1. Specification

- Use the modular language COD, which allows us to separately specify
  - processes (as Communicating Sequential Processes, CSP),
  - data (using Object-Z, OZ), and
  - time (using the Duration Calculus, DC).

2. Verification

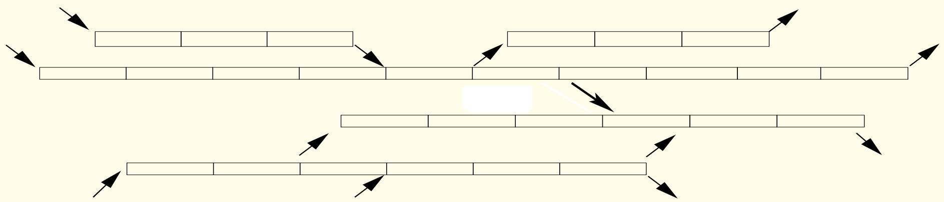
- **Verification tasks: invariant checking.**
  - $\mapsto$ Problem: reasoning in complex data structures
  - $\mapsto$ Solution: hierarchical and modular reasoning
- **Use of COD allows us to decouple:**
  - $\mapsto$ Verification tasks concerning data (OZ)
  - $\mapsto$ Verification tasks concerning durations (DC)

Allows us to impose/verify conditions on the single components which guarantee safety of the overall system.

Main goal: exploit modularity at various levels

3. Structurally

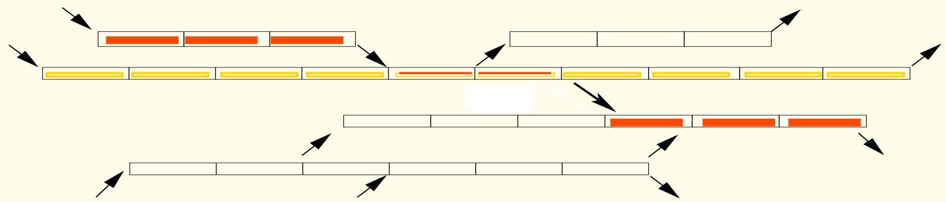
- **Running example:** Complex track topologies



Main goal: exploit modularity at various levels

3. Structurally

- **Running example:** Complex track topologies



$\mapsto$ One line track: Verification

$\mapsto$ Complex track topology:

- decomposition into family of linear tracks
- prove that safety of whole system follows from safety for the controller of a linear track.

Overview

- Modular Specifications: COD
- Modular Verification
- Modularity at structural level
- Implementation; experimental results
- Conclusions

Overview

- Modular Specifications: COD
- Modular Verification
- Modularity at structural level
- Implementation; experimental results
- Conclusions

Modular Specifications: CSP-OZ-DC (COD)

COD [Hoenicke,Olderog'02] allows us to specify in a modular way:

- the control flow of a system
  using Communicating Sequential Processes (CSP)
- the state space and its change
  using Object-Z (OZ)
- (dense) real-time constraints over durations of events
  using the Duration Calculus (DC)

Modular Specifications: CSP-OZ-DC (COD)

COD [Hoenicke,Olderog'02] allows us to specify in a modular way:

- the control flow of a system
  using Communicating Sequential Processes (CSP)
- the state space and its change
  using Object-Z (OZ)
- (dense) real-time constraints over durations of events
  using the Duration Calculus (DC)

Benefits:

- Compositionality: it suffices to prove safety properties for the separate components to prove safety of the entire system
- high-level tool support given by Syspect (easy-to-use front-end to formal real-time specifications, with a graphical user interface).

Example: Controller for line track (RBC)

RBC

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

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

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

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

CSP

SegmentData

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

TrainData

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

$sd : SegmentData$
$td : TrainData$

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

Init

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

OZ

$\texttt{effect\_}updSpd$

$\Delta(spd)$

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

$\vdots$

Interface

CSP part

Data classes

State and Init schema

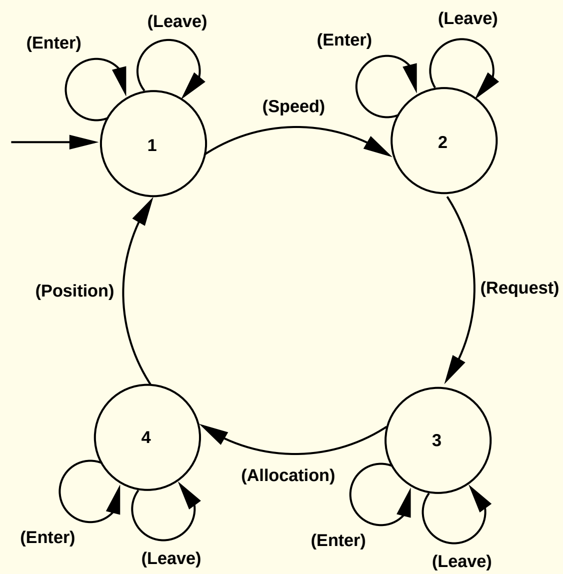
Update rules

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:

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

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

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

$$\texttt{main} \stackrel{c}{=} ((updSpd \to State1) \ \Box (leave \to \texttt{main}) \ \Box (enter \to \texttt{main}))$$

$$State1 \stackrel{c}{=} ((req \to State2) \ \Box (leave \to State1) \ \Box (enter \to State1))$$

$$State2 \stackrel{c}{=} ((alloc \to State3) \ \Box (leave \to State2) \ \Box (enter \to State2))$$

$$State3 \stackrel{c}{=} ((updPos \to \texttt{main}) \ \Box (leave \to State3) \ \Box (enter \to State3))$$

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.

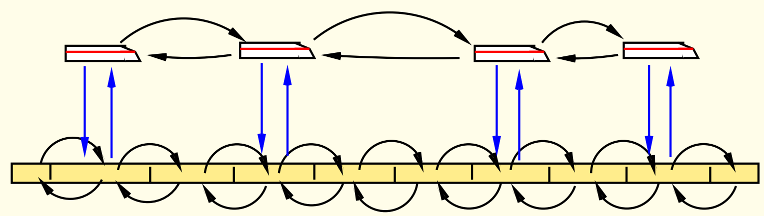
- 1. Data classes declare function symbols that can change their values during runs of the system

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

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

Timed train controller (Train)

Train consists of three timed components running in parallel.

1. Update the train's position.

   This component contains DC formulae of the form:

   $$\neg(\mathit{true}\,;\updownarrow \mathit{updPos}\,;(\ell < \Delta t)\,;\updownarrow \mathit{updPos}\,;\mathit{true}),$$

   $$\neg(\mathit{true}\,;\updownarrow \mathit{updPos}\,;(\ell > \; c \;)\,;\updownarrow \mathit{updPos}\,;\mathit{true}),$$

   that specify lower/upper time bounds on *updPos* events.

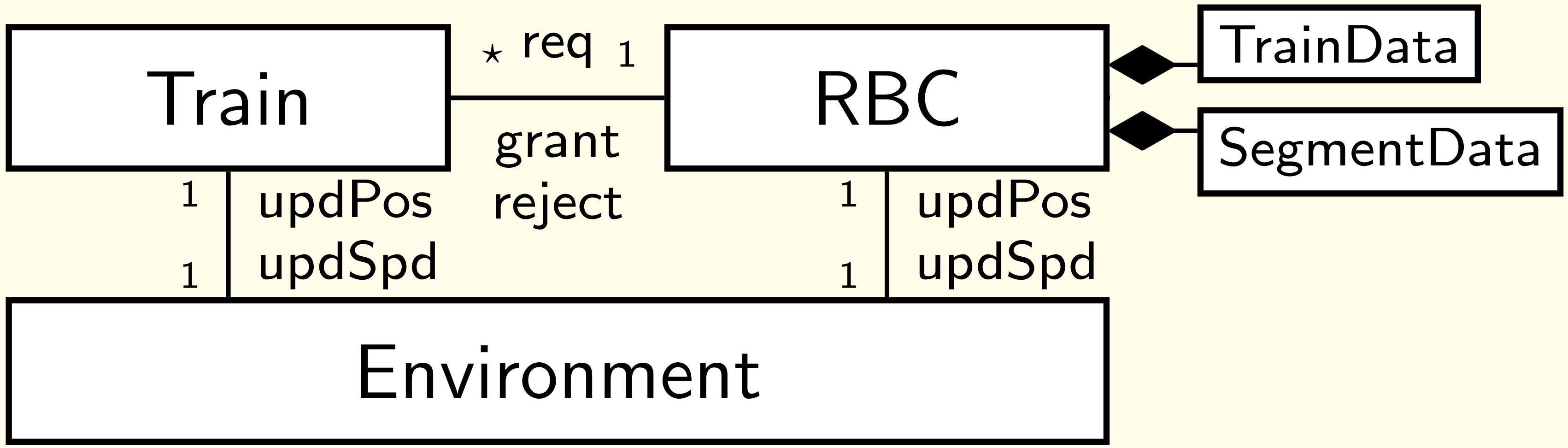
2. Check if train is beyond the safety distance to the end of the segment.

   If so, it starts braking within a short reaction time.

3. Request extension of the movement authority from the RBC

   (may be granted or rejected).

Interaction RBC/Train



Overview

- Modular Specifications: CSP-OZ-DC
- Modular Verification
- Modularity at structural level
- Implementation; experimental results
- Conclusions

Modular Verification

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

Given: Safe(x) formula (e.g. safety property)

- **Invariant checking**

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

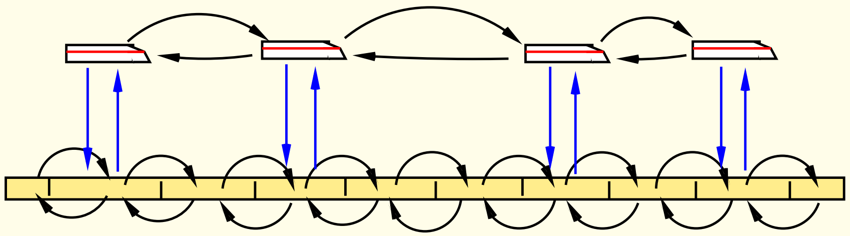
  (2) $\models_{\mathcal{T}_S}$ Safe($\overline{x}$)$\wedge$Update($\overline{x}, \overline{x}'$)$\rightarrow$Safe($\overline{x}'$) (Safe holds before $\Rightarrow$ holds after update)

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

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

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

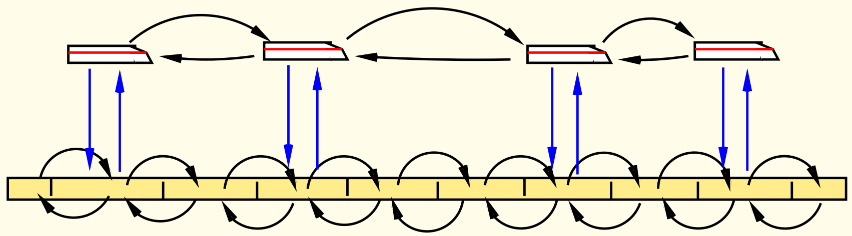
Trains on a linear track



Example 1: Speed Update

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

Trains on a linear track



Example 1: Speed Update

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

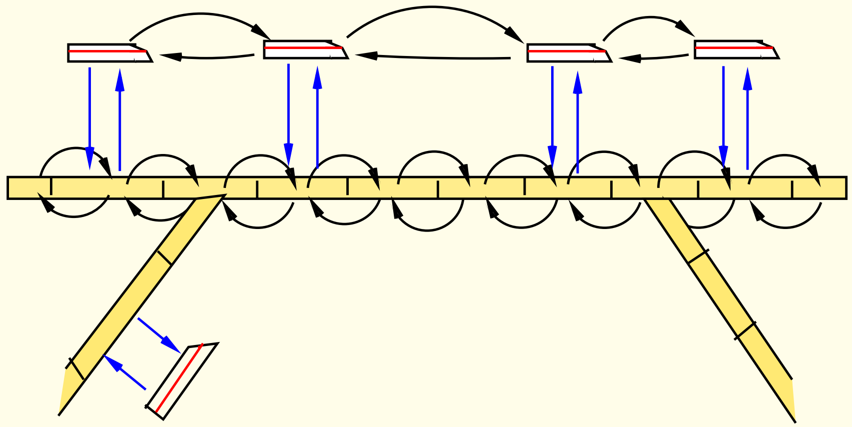
$$\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d \;\wedge\; \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) = \mathsf{tid}(t)$$
$$\rightarrow\; 0 \leq \mathsf{spd}'(t) \leq \min(\mathsf{lmax}(\mathsf{segm}(t)), \mathsf{lmax}(\mathsf{next}_s(\mathsf{segm}(t))))$$

$$\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d \;\wedge\; \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) \neq \mathsf{tid}(t)$$
$$\rightarrow\; \mathsf{spd}'(t) = \max(\mathsf{spd}(t) - \mathsf{decmax}, 0)$$

Proof task:

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

Incoming and outgoing trains



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

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

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

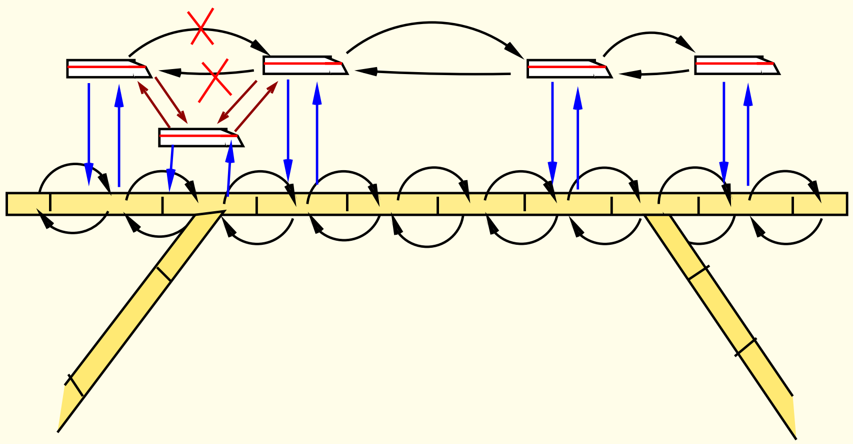
$t{\neq}t_1, \mathrm{ids}(\mathrm{segm}(t)){<}\mathrm{ids}(s_1), \mathrm{alloc}(s_1){=}\mathrm{tid}(t_1), \mathrm{next}_t(t){\neq}\mathrm{null}_t, \mathrm{ids}(\mathrm{segm}(\mathrm{next}_t(t))){\leq}\mathrm{ids}(s_1)$

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

...

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

Incoming and outgoing trains



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

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

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

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

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

...

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

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) (Inv_i) $\models$ (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:

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

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

(2) the invariants are preserved under all transitions of the system,
(Inv_i) $\wedge$ (Update) $\models$ (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**

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

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**

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

$$\dots$$

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

Modularity in automated reasoning

Examples of theories we need to handle

- **Invariants**

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

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

- **Update rules**

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

$$\ldots$$

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

- **Underlying theory:** theory of many-sorted pointers, real numbers, ...

Local theory extensions

Our approach: Find complete instantiations of univ. quantified variables

[VS'05] $\Sigma_0 \subseteq \Sigma_0 \cup \Sigma$; $\mathcal{K}$ clauses axiomatizing functions in Σ; $\mathcal{T}_0$ Σ_0-theory;

(Loc) $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is local, if for any (finite) set of ground clauses G,

$$
\begin{array}{ccccc}
\mathcal{T}_0 \cup \mathcal{K} \cup G \models \perp & \text{iff} & \mathcal{T}_0 \cup \mathcal{K}[G] \cup G \models \perp & & \\
 & \Leftarrow & & & \text{always} \\
 & \Rightarrow & & & \text{locality}
\end{array}
$$

Various notions of locality, depending of the instances to be considered closure operator on ground terms: [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10]

Local theory extensions

Our approach: Find complete instantiations of univ. quantified variables

[VS'05] $\Sigma_0 \subseteq \Sigma_0 \cup \Sigma$; $\mathcal{K}$ clauses axiomatizing functions in Σ; $\mathcal{T}_0$ Σ_0-theory;

(Loc) $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is local, if for any (finite) set of ground clauses G,

$$\mathcal{T}_0 \cup \mathcal{K} \cup G \models \perp \quad \text{iff} \quad \mathcal{T}_0 \cup \mathcal{K}[G] \cup G \models \perp$$

$\Leftarrow$ always

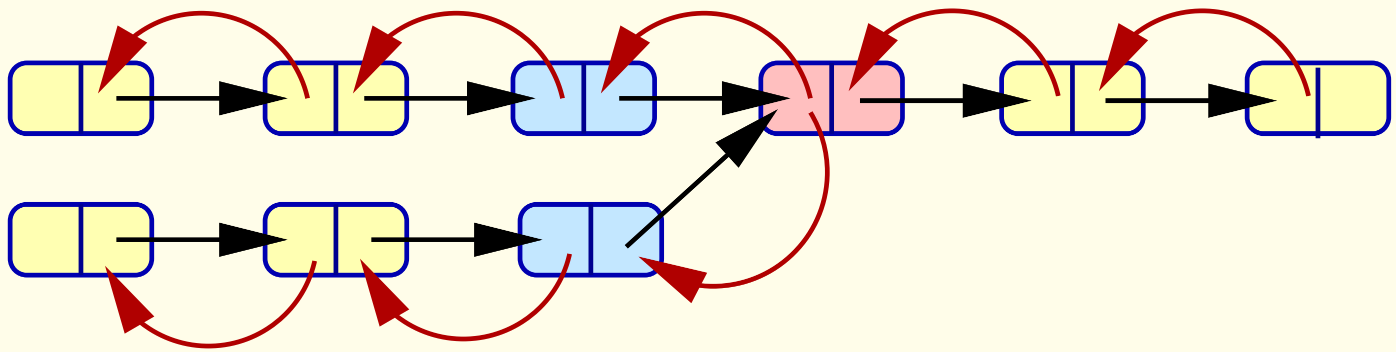
$\Rightarrow$ locality

Various notions of locality, depending of the instances to be considered
closure operator on ground terms: [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10]

Main advantages:

- $\mapsto$ hierarchical reduction to proof tasks in $\mathcal{T}_0$
- $\mapsto$ decision procedure for satisfiability of ground clauses
- $\mapsto$ implementation H-PILoT [Ihlemann,VS'2009]

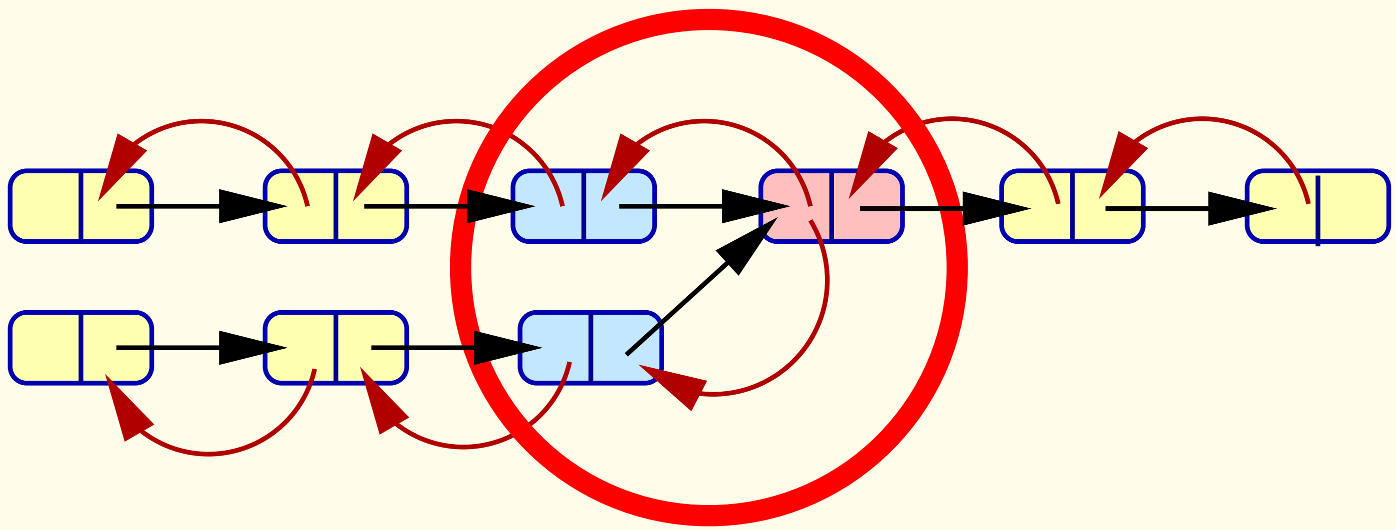
Example: doubly-linked lists



$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow \ p.\text{next}.\text{prev} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{prev} \neq \text{null} \ \rightarrow \ p.\text{prev}.\text{next} = p)$$

$$\wedge \quad c{\neq}\text{null} \wedge c.\text{next}{\neq}\text{null} \wedge d{\neq}\text{null} \wedge d.\text{next}{\neq}\text{null} \wedge c.\text{next}{=}d.\text{next} \wedge c \neq d \quad \models \quad \bot$$

Example: doubly-linked lists



$(c \neq \mathrm{null} \wedge c.\mathrm{next} \neq \mathrm{null} \rightarrow c.\mathrm{next}.\mathrm{prev} = c)$ $\quad (c.\mathrm{next} \neq \mathrm{null} \wedge c.\mathrm{next}.\mathrm{next} \neq \mathrm{null} \rightarrow c.\mathrm{next}.\mathrm{next}.\mathrm{prev} = c.\mathrm{next})$

$(d \neq \mathrm{null} \wedge d.\mathrm{next} \neq \mathrm{null} \rightarrow d.\mathrm{next}.\mathrm{prev} = d)$ $\quad (d.\mathrm{next} \neq \mathrm{null} \wedge d.\mathrm{next}.\mathrm{next} \neq \mathrm{null} \rightarrow d.\mathrm{next}.\mathrm{next}.\mathrm{prev} = d.\mathrm{next})$

$\wedge \quad c \neq \mathrm{null} \wedge c.\mathrm{next} \neq \mathrm{null} \wedge d \neq \mathrm{null} \wedge d.\mathrm{next} \neq \mathrm{null} \wedge c.\mathrm{next} = d.\mathrm{next} \wedge c \neq d \quad \models \quad \perp$

Similar results also if numerical info is stored in list

The good news

The following sets of formulae define local theory extensions:

- Updates (according to a partition of the state space)
- The invariants we consider
- The axioms for many-sorted pointer structures we consider

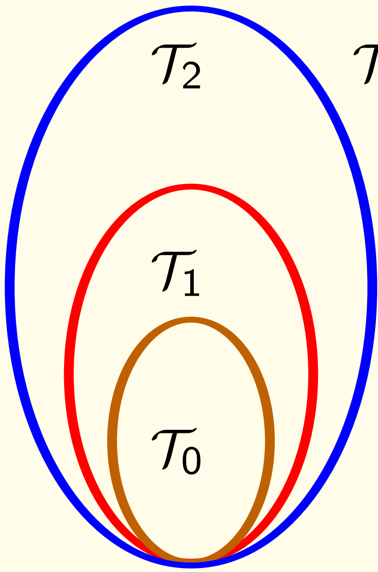
The good news

The following sets of formulae define local theory extensions:

- Updates (according to a partition of the state space)
- The invariants we consider
- The axioms for many-sorted pointer structures we consider

To show:

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



$UIF \cup \mathbb{R}$

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

$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Inv}(\mathsf{next}, \ldots)$$

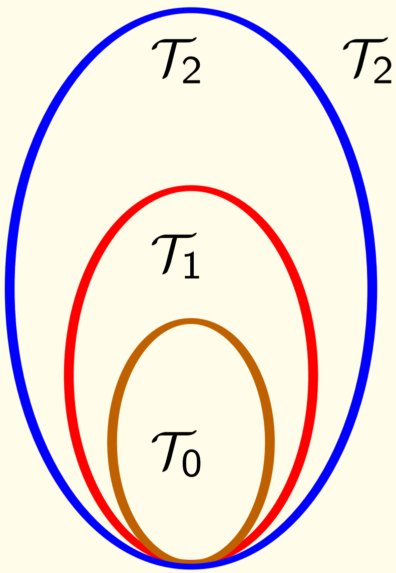
$$\mathcal{T}_0 = (\mathsf{Pointers}, \mathbb{R})$$

The good news

The following sets of formulae define local theory extensions:

- Updates (according to a partition of the state space)
- The invariants we consider
- The axioms for many-sorted pointer structures we consider

To show:



$UIF \cup \mathbb{R}$

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

$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Inv}(\mathsf{next}, ...)$$

$$\mathcal{T}_0 = (\mathsf{Pointers}, \mathbb{R})$$

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

$$\Downarrow$$

$$\mathcal{T}_1 \cup \underbrace{\mathsf{Update}[G] \wedge G}_{G'} \models \perp$$

$$\Downarrow$$

$$\mathcal{T}_0 \cup \underbrace{\mathsf{Inv}[G'] \wedge G'}_{G''} \models \perp$$

$$\Downarrow$$

$$UIF \cup \mathbb{R} \cup (\mathsf{PointerAx}[G''] \cup G'')_0 \models \perp$$

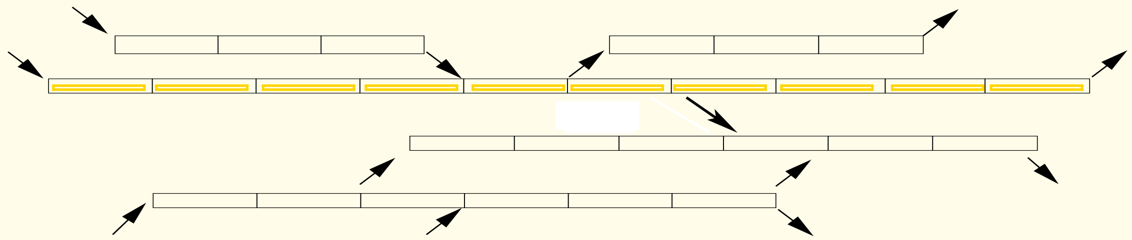
H-PILoT: verification/ models/QE $\mapsto$ constraints on parameters

Overview

- Modular Specifications: CSP-OZ-DC
- Modular Verification
- Modularity at structural level
- Implementation; experimental results
- Conclusions

Modularity at structural level

- **Complex track topologies**



Assumptions:

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

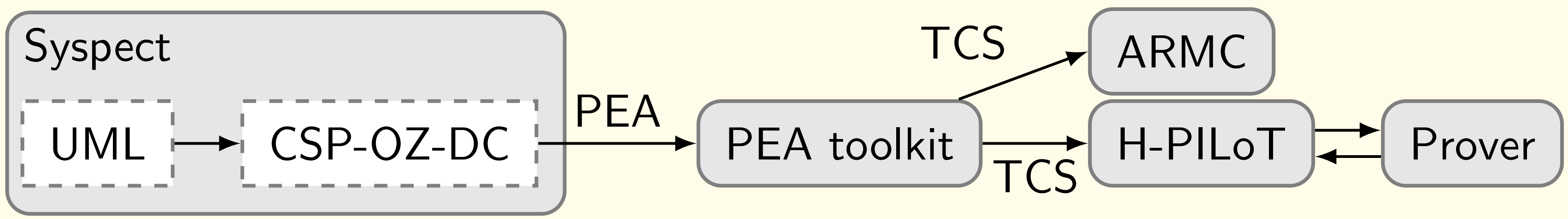
Approach:

- 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

Overview

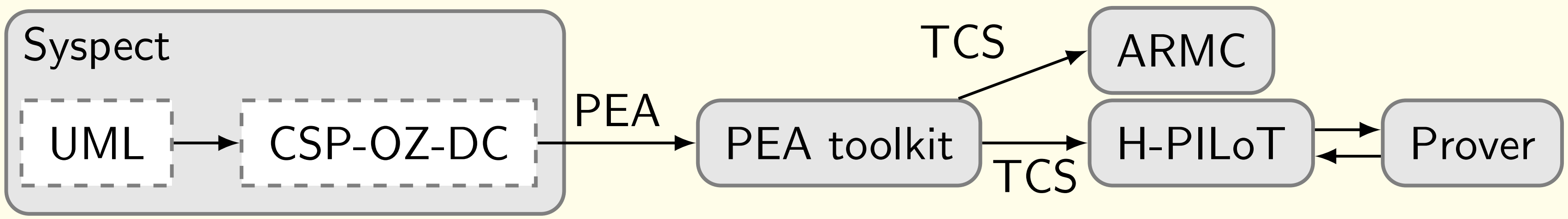
- Modular Specifications: CSP-OZ-DC
- Modular Verification
- Modularity at structural level
- Implementation; experimental results
- Conclusions

Tool Chain



Syspect
UML
CSP-OZ-DC
PEA
PEA toolkit
TCS
ARMC
TCS
H-PILoT
Prover

Experimental results



Verification of **RBC**	(Syspect + PEA)	(H-PILoT + Yices)	(Yices alone)
(Inv) *unsat*			
Part 1	11s	72s	52s
Part 2	11s	124s	131s
speed update	11s	8s	45s
(Safe) *sat*	9s	8s (+ model)	time out
Consistency	13s	3s	(Unknown) 2s

(obtained on: AMD64, dual-core 2 GHz, 4 GB RAM)

Verification of **Train:** 8 parallel components, $>$ 3300 transitions,
28 real-valued variables, clocks (infinite state system).
For this reason, the verification took 26 hours

Summary

Main approach: Exploit modularity in specification/verification/structure

Contributions: [Faber, Ihlemann, Jacobs, VS, 2010]

- We augmented existing techniques for the verification of real-time systems to cope with rich data structures like pointer structures (and identified a decidable fragment of this theory).
- We established various modularity results.
- We implemented our approach in a new tool chain taking high-level specifications in terms of COD as input.

Beyond Yes/No

We consider parametric systems

– parametric data, parametric change, parametric environment (functions)

– parametric topology of the system (data structures)

Given: Safety property (formula Φ)

Task: 1. Check if constraints on parameters guarantee safety

2. Infer relationships between parameters,
resp. properties of the functions modeling the changes
which ensure that the safety property Φ is an invariant

3. Find models (situations when safety property does not hold)

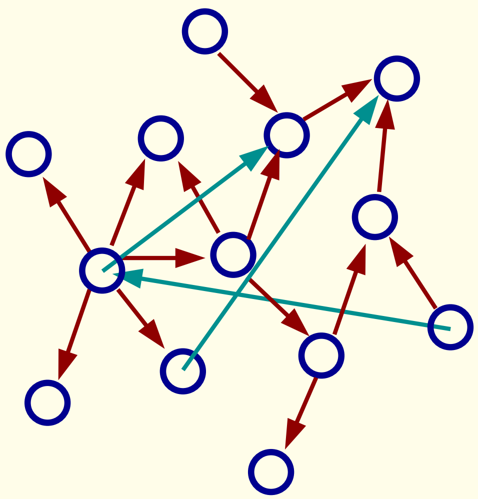
[VS; IJCAR'10) and [VS: CADE'13)

- Use the "good" properties of theories occurring in verification
- Exploit possibilities for

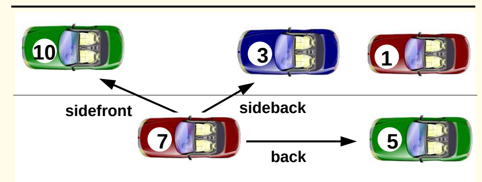
' hierarchical reasoning (1), quantifier elimination (2), model building (3)

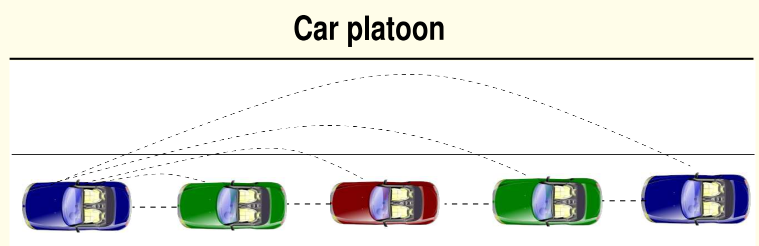
Further extensions

[Damm, Horbach, VS: FroCoS'15] Modularity results and small model property results for (decoupled) families of linear hybrid automata



Examples:





Sensors + Communication Channels

Safety properties: $\forall i_1, \ldots, i_k \;\; \phi_{\mathrm{safe}}(i_1, \ldots, i_l)$

Collision free: $\forall i, j(\mathrm{lane}(i){=}\mathrm{lane}(j) \wedge \mathrm{pos}(i){\geq}\mathrm{pos}(j) \wedge i{\neq}j \rightarrow \mathrm{pos}(i){-}\mathrm{pos}(j){>}d)$

Conclusions

Main approach: Exploit modularity in specification/verification/structure

Application areas:

- Verification of real time systems [Faber,Ihlemann,Jacobs,VS'10]
- Verification of hybrid systems [Damm,Horbach,VS'15]

Main idea:

- Use locality of the decidable fragment of the theory of pointers and of updates to simplify verification tasks.
- By-product: Small model property, complexity estimation
- Parametric verification and model building possible

Implementations

- Chain tool for real time systems
- Verification tool for families of LHA

Conclusions

Main approach: Exploit modularity in specification/verification/structure

Application areas:

- Verification of real time systems [Faber,Ihlemann,Jacobs,VS'10]
- Verification of hybrid systems [Damm,Horbach,VS'15]

Main idea:

- Use locality of the decidable fragment of the theory of pointers and of updates to simplify verification tasks.
- By-product: Small model property, complexity estimation
- Parametric verification and model building possible

Ongoing and future work: More complex combinations/properties
– Time-bounded reachability conditions (e.g. overtaking manoeuvers)
– Invariant generation