Modularity in automated reasoning and in the verification of complex systems

Viorica Sofronie-Stokkermans

University Koblenz-Landau and
Max-Planck-Institut für Informatik, Saarbrücken

IFIP WG 2.2 meeting, Lisbon, September 23-26, 2013

Motivation

Goal: use computers as "intelligent assistants"
in mathematics, verification, engineering, databases ...

Main problem: complexity

- complex description of problems to be solved
  - $\mapsto$ complex encoding
  - $\mapsto$ large formulae
- system dynamics
- complex systems (interaction, synchronization)

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

- **Theories from mathematical analysis**

  Functions over $\mathbb{R}$

  - monotone, bounded
  - continuous, differentiable

- **Algebraic structures**

  Monoids, groups, rings

  Lattices, Boolean algebras

- **Logic**

  Classical logic

  Non-classical logics
  - many-valued logics, fuzzy logic
  - modal, dynamic, temporal
  - logics for MAS

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- programs
correctness/termination
- reactive/hybrid systems
safety/lifeness
- cryptography
correctness crypt. prot.

Theories
- numbers
- data types
- functions over numeric domains

Controllers

Embedded software

Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks

- programs

correctness/termination

- reactive/hybrid systems

safety/lifeness

- cryptography

correctness crypt. p

Theories

- numbers
- data types
- functions ove numeric doma

Controllers

Embedded software

Program verification

```
int [] BubbleSort(int[] a) {
   int i, j, t;
   for (i :=| a | −1;  i > 0;  i := i − 1) {
     for (j := 0;  j < i;  j := j + 1) {
       if (a[j] > a[j + 1]){t := a[j];
                            a[j] := a[j + 1];
                            a[j + 1] := t};
}} return a}
```

- Does BubbleSort return a sorted array?
- Is a state with a certain property reachable in $\leq k$ steps?

Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

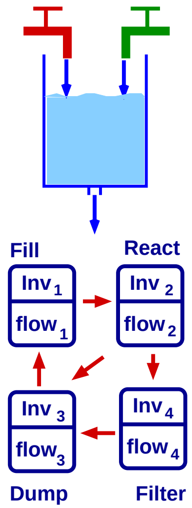
Tasks

- programs
  correctness/termination
- reactive/hybrid systems
  safety/lifeness
- cryptography
  correctness crypt. prot.

Theories

- numbers
- data types
- functions over numeric domains

Controllers



Check:

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

Determine values for parameters such that this is the case

Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks

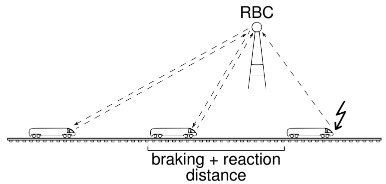
- programs
  correctness/termination
- reactive/hybrid systems
  safety/lifeness
- cryptography
  correctness crypt. prot.

Theories

- numbers
- data types
- functions over numeric domains

Controllers

Train controllers



- **Task:** check collision freeness

Examples of application domains

MATHEMATICS

Tasks
- construct proofs
- check proofs

Theories
- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks
- programs

correctness/termination
- reactive/hybrid systems

safety/lifeness
- cryptography

correctness crypt. prot.

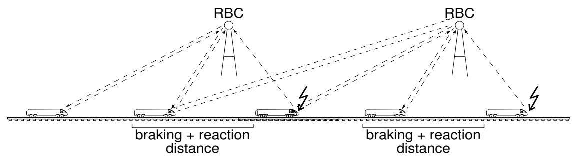
Theories
- numbers
- data types
- functions over numeric domains

Controllers

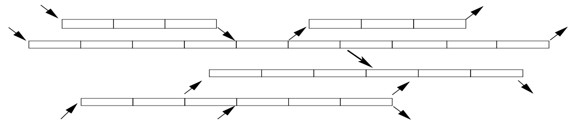


Two or more controllers

- non-disjoint sets of controlled trains
- synchronization for the control of common trains



- complex track topology



Examples of application domains

MATHEMATICS

Tasks

- construct proofs
- check proofs

Theories

- numbers
- polynomials
- functions over numeric domains
- algebras

VERIFICATION

Tasks

- programs

correctness/terminati

- reactive/hybrid systems

safety/lifeness

- cryptography

correctness crypt. p

Theories

- numbers
- data types
- functions ove numeric domains

Verification tasks

can often be solved as follows:

- encode problems as logical formulae
- reduce verification tasks to testing entailment/satisfiability/validity

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

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

The goals of my research:

- Identify decidable theories which are important in applications (extensions/combinations) possibly with low complexity
- Development & implementation of efficient decision procedures
- Applications, e.g. in verification, databases, mathematics

Goal of this talk

Present some of my work on identifying conditions under which efficient methods for the verification of complex systems exist

Focus: Modularity

- Modularity in automated reasoning (Application: Verification)

  Deductive verification

  Synthesis: Generate constraints on parameters which guarantee satisfiability

- Modularity in the verification of complex, interacting systems

Goal of this talk

Present some of my work on identifying conditions
under which efficient methods for the verification
of complex systems exist

Structure of the talk

- Efficient automated reasoning: hierarchical and modular reasoning
- Local theory extensions (idea: complete instantiation)
- Recognizing local theory extensions; examples
- Applications

  Deductive verification (invariant checking, BMC)

  Synthesis: Generate constraints on parameters which guarantee satisfiability
- Example: Modular verification - system of trains; complex track topology

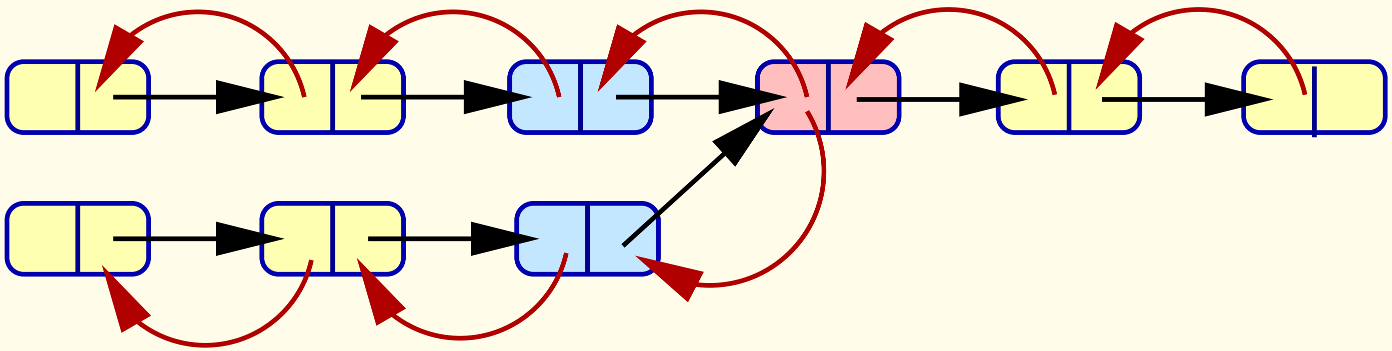
Automated reasoning

Important for efficient reasoning

- Possibility of limiting search
- Modular reasoning in complex theories

Example: A theory of doubly-linked lists

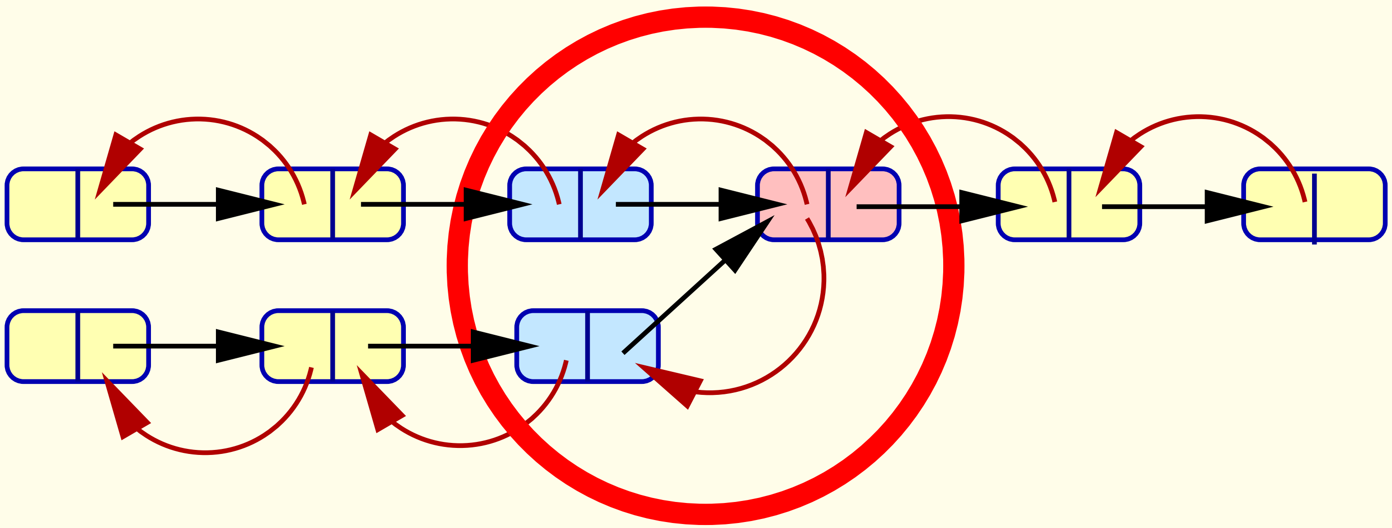
[Necula, McPeak, 2005]



$$\forall p\ (p \neq \text{null} \wedge p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{next}.\text{prev} = p)$$
$$\forall p\ (p \neq \text{null} \wedge p.\text{prev} \neq \text{null} \ \rightarrow\ p.\text{prev}.\text{next} = p)$$
$$\wedge\ \ c{\neq}\text{null} \wedge c.\text{next}{\neq}\text{null} \wedge d{\neq}\text{null} \wedge d.\text{next}{\neq}\text{null} \wedge c.\text{next}{=}d.\text{next} \wedge c \neq d \quad \models \quad \perp$$

Example: A theory of doubly-linked lists

[Necula, McPeak, 2005]



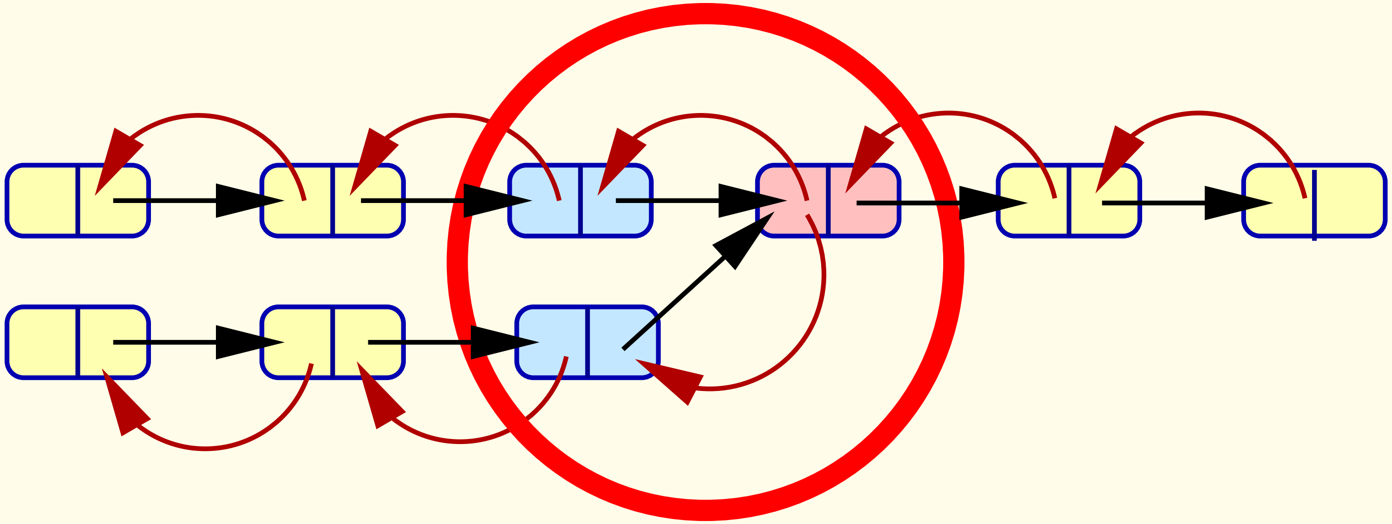
$(c \neq \text{null} \wedge c.\text{next} \neq \text{null} \rightarrow c.\text{next}.\text{prev} = c) \quad (c.\text{next} \neq \text{null} \wedge c.\text{next}.\text{next} \neq \text{null} \rightarrow c.\text{next}.\text{next}.\text{prev} = c.\text{next})$

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

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

Example: A theory of doubly-linked lists

[Necula, McPeak, 2005]



($c\neq$null ... c.next)

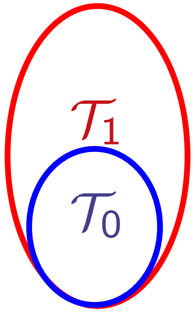
($d\neq$null ... d.next)

$\wedge$

Consider extensions which also take the **elements** of the list into account

$\mapsto$ Reasoning in complex theories

Complex Theories



Hierarchic Reasoning

$\mathcal{T}_1$: Σ_1-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_1$ $\Sigma_0 \subset \Sigma_1$

$\mathcal{T}_0$: Σ_0-theory.

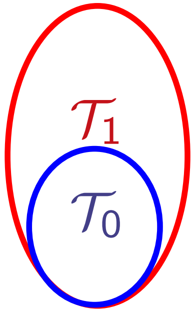
Example:

$f : \mathbb{R} \rightarrow \mathbb{R}$ mon.

$\mathbb{R}$

Can we use a prover for $\mathcal{T}_0$ as a blackbox to prove theorems in $\mathcal{T}_1$?

Complex Theories



Hierarchic Reasoning

$\mathcal{T}_1$: Σ_1-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_1$ $\Sigma_0 \subset \Sigma_1$

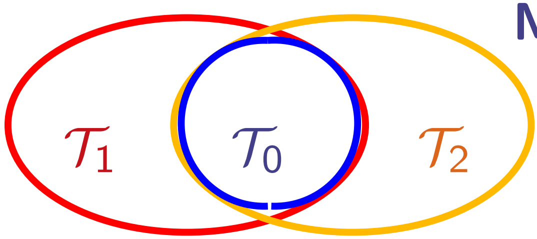
$\mathcal{T}_0$: Σ_0-theory.

Example:

$f : \mathbb{R} \rightarrow \mathbb{R}$ mon.

$\mathbb{R}$

Can we use a prover for $\mathcal{T}_0$ as a blackbox to prove theorems in $\mathcal{T}_1$?



Modular Reasoning

$\mathcal{T}_0$: Σ_0-theory.

$\mathcal{T}_i$: Σ_i-theory; $\mathcal{T}_0 \subseteq \mathcal{T}_i$ $\Sigma_0 \subseteq \Sigma_i$.

Example:

lists($\mathbb{R}$) $\cup$ **arrays**($\mathbb{R}$)

Can we use provers for $\mathcal{T}_1, \mathcal{T}_2$ as blackboxes to prove theorems in $\mathcal{T}_1 \cup \mathcal{T}_2$?
Which information needs to be exchanged between the provers?

Example

$$\mathbb{R} \cup \mathsf{Mon}_f \wedge \underbrace{(a < b \wedge f(a) = f(b) + 1)}_{G} \models \perp$$

$\mathsf{Mon}_f \qquad \forall x, y (x \leq y \rightarrow f(x) \leq f(y))$

- A prover for $\mathbb{R}$ cannot handle the function f
- A prover for first-order logic cannot handle real numbers

Idea: Hierarchical reasoning

Step 1: Reasoning about the properties of f

Step 2: Reasoning about real numbers

Idea

$$\mathbb{R} \cup \mathrm{Mon}_f \wedge \underbrace{(a < b \wedge f(a) = f(b) + 1)}_{G} \models \perp$$

Limit search space

$$\mathbb{R} \cup \mathrm{Mon}_f[G] \cup G \models \perp$$

$\mapsto$ sound and complete

	$G \cup \mathrm{Mon}(f)$
	$a < b$
	$f(a) = f(b) + 1$
	$\forall x, y(x \leq y \rightarrow f(x) \leq f(y))$

Idea

$$\mathbb{R} \cup \mathrm{Mon}_f \wedge \underbrace{(a < b \wedge f(a) = f(b) + 1)}_{G} \models \perp$$

Limit search space

$$\mathbb{R} \cup \mathrm{Mon}_f[G] \cup G \models \perp$$

$\mapsto$ sound and complete

Hierarchical reasoning

The following are equivalent:

(1) $\mathbb{R} \cup \mathrm{Mon}_f[G] \cup G \models \perp$

(2) $\mathbb{R} \cup \mathrm{Mon}_f[G]_0 \cup G_0 \cup \mathrm{Def} \models \perp$ (Purification)

	$G \cup \mathrm{Mon}(f)[G]$
$a_1 = f(a)$	$a < b$
$b_1 = f(b)$	$f(a) = f(b) + 1$
	$a \leq b \rightarrow f(a) \leq f(b)$
	$b \leq a \rightarrow f(b) \leq f(a)$

Idea

$$\mathbb{R} \cup \mathsf{Mon}_f \wedge \underbrace{(a < b \wedge f(a) = f(b) + 1)}_{G} \models \perp$$

Definitions	$G_0 \cup \mathsf{Mon}(f)[G]_0 \cup \mathsf{Con}[G]_0$
$a_1 = f(a)$	$a < b$
$b_1 = f(b)$	$a_1 = b_1 + 1$
	$a \leq b \rightarrow a_1 \leq b_1$
	$b \leq a \rightarrow b_1 \leq a_1$
	$a = b \rightarrow a_1 = b_1$

Limit search space

$$\mathbb{R} \cup \mathsf{Mon}_f[G] \cup G \models \perp$$

$\mapsto$ sound and complete

Hierarchical reasoning

The following are equivalent:

(1) $\mathbb{R} \cup \mathsf{Mon}_f[G] \cup G \models \perp$

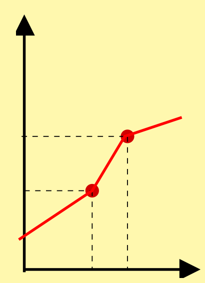
(2) $\mathbb{R} \cup \mathsf{Mon}_f[G]_0 \cup G_0 \cup \mathsf{Def} \models \perp$ (Purification)

(3) $\mathbb{R} \cup \mathsf{Mon}_f[G]_0 \cup G_0 \cup \mathsf{Con}(\mathsf{Def}) \models \perp$ (Hierarchical reduction)

Idea

$$\mathbb{R} \cup \mathrm{Mon}_f \wedge \underbrace{(a < b \wedge f(a) = f(b) + 1)}_{G} \models \perp$$

Definitions	$G_0 \cup \mathrm{Mon}(f)[G]_0 \cup \mathrm{Con}[G]_0$
$a_1 = f(a)$	$a < b$
$b_1 = f(b)$	$a_1 = b_1 + 1$
	$a \leq b \rightarrow a_1 \leq b_1$
	$b \leq a \rightarrow b_1 \leq a_1$
	$a = b \rightarrow a_1 = b_1$



Limit search space

$$\mathbb{R} \cup \mathrm{Mon}_f[G] \cup G \models \perp$$

$\mapsto$ sound and complete

Hierarchical reasoning

The following are equivalent:

(1) $\mathbb{R} \cup \mathrm{Mon}_f[G] \cup G \models \perp$

(2) $\mathbb{R} \cup \mathrm{Mon}_f[G]_0 \cup G_0 \cup \mathrm{Def} \models \perp$ (Purification)

(3) $\mathbb{R} \cup \mathrm{Mon}_f[G]_0 \cup G_0 \cup \mathrm{Con}(\mathrm{Def}) \models \perp$ (Hierarchical reduction)

Local theory extensions

[GSW'04, VS'05] $\mathcal{K}$ set of equational clauses; $\mathcal{T}_0$ theory; $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$

Definition. $\mathcal{T}_0 \subseteq \mathcal{T}_1$ is local iff for all sets of ground clauses G,

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

Extends the notion of local theory introduced in [Givan,McAllester'92,'94]

and further studied in [BasinGanzinger'96,'01, Ganzinger'01]

Local theory extensions

[GSW'04, VS'05] $\mathcal{K}$ set of equational clauses; $\mathcal{T}_0$ theory; $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$

Definition. $\mathcal{T}_0 \subseteq \mathcal{T}_1$ is local iff for all sets of ground clauses G,

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

Hierarchical reasoning possible [VS'05]

1. **Locality:** $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G \models \perp$ $\mapsto$ $\mathcal{O}(n^k)$ clauses
2. **Purification:** $\mathcal{T}_0 \cup \mathcal{K}[G]_0 \cup G_0 \cup \text{Def} \models \perp$ $\mapsto$ linear
3. **Hierarchical reduction:** $\mathcal{T}_0 \cup \mathcal{K}[G]_0 \cup G_0 \cup \text{Con}(\text{Def}) \models \perp$ $\mapsto$ $+\mathcal{O}(n^2)$ clauses
4. Satisfiability test in $\mathcal{T}_0$ (prover for $\mathcal{T}_0$ – blackbox) $\mapsto$ $g(n^k)$

Parametric complexity for $\mathcal{T}_1$

Local theory extensions

Various notions of locality, depending of the instances to be considered (closure operators [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10])

Implementation: H-PILoT [Ihlemann,VS'09]

How to recognize local theory extensions?

- Embeddability of partial into total models [Ganzinger,VS,Waldmann'04,VS'05]
- Saturation (under resolution) [Basin,Ganzinger'96'01, VS'07, Horbach,VS'13]
- Transfer of locality [VS'07, Ihlemann,VS'10]

Local theory extensions

Various notions of locality, depending of the instances to be considered (closure operators [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10])

Implementation: H-PILoT [Ihlemann,VS'09]

How to recognize local theory extensions?

- Embeddability of partial into total models [Ganzinger,VS,Waldmann'04,VS'05]

  $T_0 \cup \mathcal{K} \cup G \models \perp$ iff $T_0 \cup \mathcal{K}[G] \cup G \models \perp$

- Saturation (under resolution) [Basin,Ganzinger'96'01, VS'07, Horbach,VS'13]

- Transfer of locality [VS'07, Ihlemann,VS'10]

Local theory extensions

Various notions of locality, depending of the instances to be considered (closure operators [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10])

Implementation: H-PILoT [Ihlemann,VS'09]

How to recognize local theory extensions?

- Embeddability of partial into total models [Ganzinger,VS,Waldmann'04,VS'05]

- Saturation (under resolution) [Basin,Ganzinger'96'01, VS'07, Horbach,VS'13]

- Transfer of locality [VS'07, Ihlemann,VS'10]

Saturation and Locality

K is order local w.r.t. $\prec$ iff for every ground clause C:

$$\mathcal{K} \models C \quad \text{if and only if} \quad \mathcal{K} \models_{\preceq} C$$

$\mathcal{K} \models_{\preceq} C$ means: there is a proof of C from those ground instances of clauses in $\mathcal{K}$ in which each term is smaller than (w.r.t. $\prec$) or equal to some term in C.

Theorem [Basin,Ganzinger'96,'01]

If $\mathcal{K}$ is reductive and saturated w.r.t. $\prec$-ordered resolution, then $\mathcal{K}$ is order local w.r.t. $\prec$.

Reductive: for each ground instance C of a clause in $\mathcal{K}$, all terms ocurring in C are smaller than or equal to some term in the maximal atom of C).

Saturation and Locality

$\mathcal{K} := \text{Pre} \cup \{y = s(x) \rightarrow f(x) \leq f(y)\}$ $\quad$ $(\text{Pre} = \{x{\leq}x, x{\leq}y \wedge y{\leq}z \rightarrow x{\leq}z\})$

When saturating this set of clauses we obtain the infinite set

$$\{y = s^n(x) \rightarrow f(x) \leq f(y) \mid n \geq 0\} \cup \text{Pre}.$$

Saturation and Locality

$\mathcal{K} := \text{Pre} \cup \{y = s(x) \rightarrow f(x) \leq f(y)\}$ $\quad$ $(\text{Pre} = \{x{\leq}x, x{\leq}y \wedge y{\leq}z \rightarrow x{\leq}z\})$

When saturating this set of clauses we obtain the infinite set

$$\{y = s^n(x) \rightarrow f(x) \leq f(y) \mid n \geq 0\} \cup \text{Pre}.$$

Our Idea [Horbach,VS 2013] Use constrained clauses: $[y = s(x)] f(x) \leq f(y)$
Develop a constrained ordered resolution calculus with melting constraints

Melting rule states: if it is possible to derive $[c[x \mapsto s(x)]] C\sigma$ from $[c] C\sigma$, then it is also possible to repeat this process to derive $[c[x \mapsto s(s(x))] C\sigma$ and so on.

Saturation and Locality

$\mathcal{K} := \text{Pre} \cup \{y = s(x) \rightarrow f(x) \leq f(y)\}$ $\quad$ $(\text{Pre} = \{x \leq x, x \leq y \wedge y \leq z \rightarrow x \leq z\})$

When saturating this set of clauses we obtain the infinite set

$$\{y = s^n(x) \rightarrow f(x) \leq f(y) \mid n \geq 0\} \cup \text{Pre}.$$

Our Idea [Horbach,VS 2013] Use constrained clauses: $[y = s(x)]f(x) \leq f(y)$
Develop a constrained ordered resolution calculus with melting constraints

Melting rule states: if it is possible to derive $[c[x \mapsto s(x)]]C\sigma$ from $[c]C\sigma$, then it is also possible to repeat this process to derive $[c[x \mapsto s(s(x))]C\sigma$ and so on. $\quad \mapsto$ finite representation (use regular expressions in constraints)

$$\mapsto \quad \{[y = s^*(x)]f(x) \leq f(y)\} \cup \text{Pre}.$$

Saturation and Locality

$\mathcal{K} := \text{Pre} \cup \{y = s(x) \rightarrow f(x) \leq f(y)\}$ $\quad$ $(\text{Pre} = \{x \leq x, x \leq y \wedge y \leq z \rightarrow x \leq z\})$

When saturating this set of clauses we obtain the infinite set

$$\{y = s^n(x) \rightarrow f(x) \leq f(y) \mid n \geq 0\} \cup \text{Pre}.$$

Our Idea [Horbach,VS 2013] Use constrained clauses: $[y = s(x)] f(x) \leq f(y)$
Develop a constrained ordered resolution calculus with melting constraints

Melting rule states: if it is possible to derive $[c[x \mapsto s(x)]] C\sigma$ from $[c] C\sigma$, then it is also possible to repeat this process to derive $[c[x \mapsto s(s(x))] C\sigma$ and so on. $\quad \mapsto$ finite representation (use regular expressions in constraints)

$$\mapsto \quad \{y = s^*(x) \rightarrow f(x) \leq f(y)\} \cup \text{Pre}.$$

Saturation and Locality

$\mathcal{K} := \text{Pre} \cup \{y = s(x) \rightarrow f(x) \leq f(y)\}$ $\quad (\text{Pre} = \{x{\leq}x, x{\leq}y \wedge y{\leq}z \rightarrow x{\leq}z\})$

When saturating this set of clauses we obtain the infinite set

$$\{y = s^n(x) \rightarrow f(x) \leq f(y) \mid n \geq 0\} \cup \text{Pre}.$$

Our Idea [Horbach,VS 2013] Use constrained clauses: $[y = s(x)]f(x) \leq f(y)$
Develop a constrained ordered resolution calculus with melting constraints

Melting rule states: if it is possible to derive $[c[x \mapsto s(x)]]C\sigma$ from $[c]C\sigma$, then it is also possible to repeat this process to derive $[c[x \mapsto s(s(x))]C\sigma$ and so on. $\mapsto$ finite representation (regular expressions in constraints; def. $\leq$)

$$\mapsto \quad \{x \leq y \rightarrow f(x) \leq f(y)\} \cup \text{Pre}.$$

Local theory extensions

Various notions of locality, depending of the instances to be considered (closure operators [Ihlemann,Jacobs,VS'08, Ihlemann,VS'10])

Implementation: H-PILoT [Ihlemann,VS'09]

How to recognize local theory extensions?

- Embeddability of partial into total models [Ganzinger,VS,Waldmann'04,VS'05]
- Saturation (under resolution) [Basin,Ganzinger'96'01, VS'07, Horbach,VS'13]
- Transfer of locality [VS'07, Ihlemann,VS'10]

  Combinations of local extensions are often local $\mapsto$ modularity

Combinations of Theories

Focus: Modularity

Combinations of theories with disjoint signature [Nelson, Oppen'79]

Combinations of theories with non-disjoint signature

- [Tinelli et al.'02–'07] Relax conditions in Nelson/Oppen proc.
- [Ghilardi et al.'03–'05] Model-theoretic method
- [Armando, Bonacina, Ranise, ...] Resolution-based methods
- [Ganzinger,VS,Waldmann'04,'06] Superposition/pure inferences
- [Ihlemann,VS'10] Combinations of local theory extensions are often local

$\mapsto$ **Interpolation** [VS'06,'08, Rybalchenko,VS'07,'10]

Examples of local theory extensions

MATHEMATICS

Tasks

− construct proofs
− check proofs

Theories

− numbers

− polynomials

− functions over numeric domains

− algebras

Extensions of a theory $\mathcal{T}_0$ with:

- free functions [VS'05]
- monotone functions [VS'05,'08], [Ihlemann,VS'07,'10]

Theories from mathematical analysis [VS'08b]

- boundedness conditions (linear combinations)
- monotone functions + bounds (linear combinations)
- bounds on derivatives (linear combinations)
- convexity/concavity
  + continuity/differentiability

Theories from algebra

- semilattices and lattices

Examples of local theory extensions

VERIFICATION

Tasks

– programs
correctness/termination

– reactive/hybrid systems
safety/lifeness

– cryptography
correctness crypt. prot.

Theories

– numbers
– data types
– functions over numeric domains

Theories of data structures [VS'07,VS'08c,Ihlemann,Jacobs,VS'08]

- fragments of the theory of "Arrays" und "Pointers"
- theories of recursive data structures + recursive functions
- "Update" axioms

Theories from mathematical analysis

- **Verification:**

Programs (data structures)
[VS'06,'07] [Ihlemann,Jacobs,VS'08]

Train systems
[Faber,Jacobs,VS'06,07],
[Faber,Ihlemann,Jacobs,VS'10]

Hybrid automata (appl: chemical plant controller)
[Damm,Ihlemann,VS'11, VS'13]

- **Security** [VS'06,'09]

Cryptography $\mapsto$ $\text{encode}(\text{decode}(x)) = x$
$\text{decode}(\text{encode}(x)) = x$

Verification

S specification	$\mapsto$	Σ_S signature of S; $\mathcal{T}_S$ theory of S
	$\mapsto$	T_S transition constraint system (TCS) defined by S
		- $Init(\overline{x})$: formula describing the initial state
		- $Tr(\overline{x}, \overline{x}')$: changes of variable values during transitions

Given: Ψ formula (e.g. safety property)

- **Invariant checking**

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

  (2) $\models_{\mathcal{T}_S} \Psi(\overline{x}) \wedge Tr(\overline{x}, \overline{x}') \rightarrow \Psi(\overline{x}')$ (If Ψ holds before it also holds after update)

- **Bounded model checking:**

  Check whether, for a fixed k, states violating Ψ are reachable by runs of T_S of length at most k, i.e. for all $0 \leq j \leq k$:

  $$Init(x_0) \wedge Tr_1(x_0, x_1) \wedge \cdots \wedge Tr_n(x_{j-1}, x_j) \wedge \neg\Psi(x_j) \models_{\mathcal{T}_S} \perp$$

Parametric verification (discrete systems)

Given: Safety property (formula Φ)

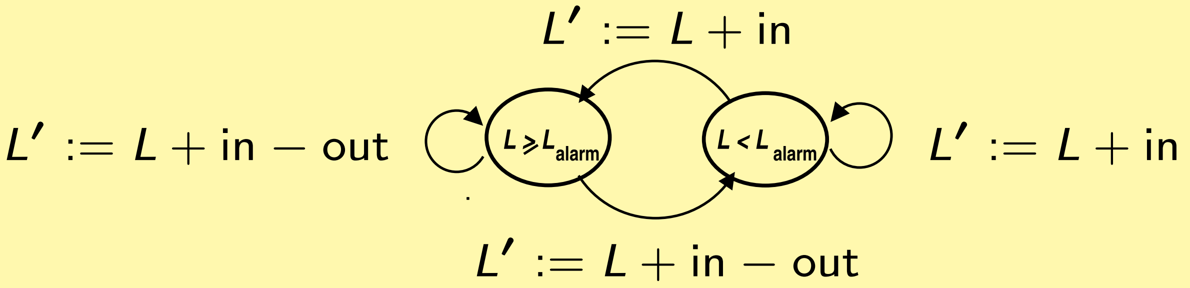
1. Verification: Check if constraints on parameters guarantee safety
If not, construct model which does not satisfy Φ.

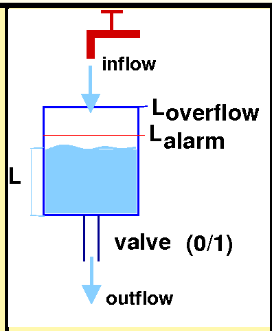
2. Synthesis: Infer relationships between parameters,
resp. properties of the functions modeling the changes
which ensure that the safety property Φ holds

Here: Invariance of safety property

Note: We used similar ideas for bounded reachability

Example 1





Initial states: $L_a \leq L \leq L_b$

Safety condition: $L \leq L_{\text{overflow}}$

Verification: Satisfiability check (for given constraints on parameters)

The following are equivalent

(1) $L \leq L_{\text{overflow}}$ is an invariant

(2) The disjunction of the following conjunctions is false

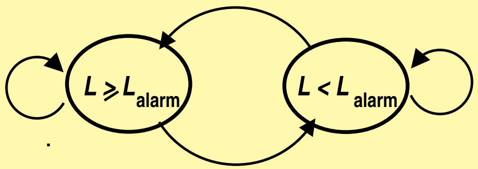
(a) $\exists L(L_a \leq L \leq L_b \wedge L > L_{\text{overflow}})$

(b)(i) $\exists L(L_{\text{alarm}} \leq L \leq L_{\text{overflow}} \wedge L + \text{in} - \text{out} > L_{\text{overflow}})$

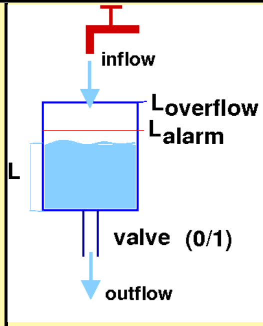
(ii) $\exists L(L < L_{\text{alarm}} \wedge L + \text{in} > L_{\text{overflow}})$

Example 1

$L' := L + \text{in}$

$L' := L + \text{in} - \text{out}$  $L' := L + \text{in}$

$L' := L + \text{in} - \text{out}$



Initial states: $L_a \leq L \leq L_b$

Safety condition: $L \leq L_{\text{overflow}}$

Verification: Satisfiability check

Synthesis: Quantifier elimination

The following are equivalent

(1) $L \leq L_{\text{overflow}}$ is an invariant

(2) The disjunction of the following conjunctions is false

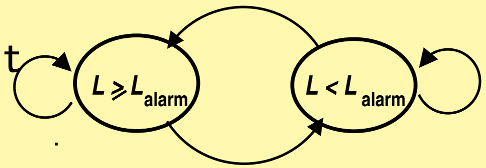
(a)	$\exists L(L_a \leq L \leq L_b \wedge L > L_{\text{overflow}})$	iff	$L_{\text{overflow}} < L_b$
(b)(i)	$\exists L(L_{\text{alarm}} \leq L \leq L_{\text{overflow}} \wedge L + \text{in} - \text{out} > L_{\text{overflow}})$	iff	$\text{in} > \text{out}$
(ii)	$\exists L(L < L_{\text{alarm}} \wedge L + \text{in} > L_{\text{overflow}})$	iff	$\text{in} > L_{\text{overflow}} - L_{\text{alarm}}$

(3) $L_b \leq L_{\text{overflow}} \ \wedge \ \text{in} \leq \text{out} \ \wedge \ \text{in} \leq L_{\text{overflow}} - L_{\text{alarm}}$

Example 2

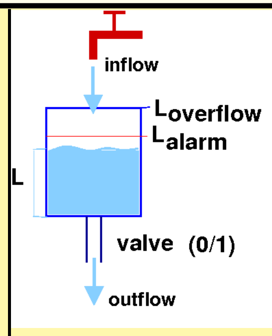
$L' := L + \textcolor{red}{in}(t),\ t' := t + 1$

$L' := L + \textcolor{red}{in}(t) - \text{out}$
$t' := t + 1$



$L' := L + \textcolor{red}{in}(t)$
$t' := t + 1$

$L' := L + \textcolor{red}{in}(t) - \text{out},\ t' := t + 1$



Initial states: $L_a \leq L \leq L_b$

Safety condition: $L \leq L_{\text{overflow}}$

Verification: Satisfiability check
Synthesis: Quantifier elimination

The following are equivalent

(1) $L \leq L_{\text{overflow}}$ is an invariant

(2) The disjunction of the following conjunctions is false

(a) $\exists L(L_a \leq L \leq L_b \wedge L > L_{\text{overflow}})$ iff $L_{\text{overflow}} < L_b$

(b)(i) $\exists L, t(L_{\text{alarm}} \leq L \leq L_{\text{overflow}} \wedge L + \textcolor{red}{in}(t) - \text{out} > L_{\text{overflow}})$

(ii) $\exists L, t(L < L_{\text{alarm}} \wedge L + \textcolor{red}{in}(t) > L_{\text{overflow}})$

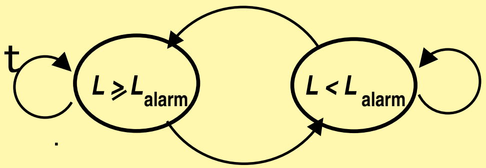
(3) $L_b \leq L_{\text{overflow}} \wedge \text{Constr}(\textcolor{red}{in})$

$\text{Constr}(\textcolor{red}{in})$: Constraints which guarantee unsatisfiability of (i),(ii)

Example 2

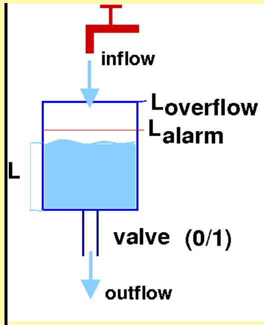
$L' := L + \mathrm{in}(t), t' := t + 1$

$L' := L + \mathrm{in}(t) - \mathrm{out}$
$t' := t + 1$



$L' := L + \mathrm{in}(t)$
$t' := t + 1$

$L' := L + \mathrm{in}(t) - \mathrm{out}, t' := t + 1$



Initial states: $L_a \leq L \leq L_b$

Safety condition: $L \leq L_{\mathrm{overflow}}$

Verification: Satisfiability check

Synthesis: Quantifier elimination

The following are equivalent

(1) $L \leq L_{\mathrm{overflow}}$ is an invariant

(2) The disjunction of the following conjunctions is false

(a) $\exists L(L_a \leq L \leq L_b \wedge L > L_{\mathrm{overflow}})$ iff $L_{\mathrm{overflow}} < L_b$

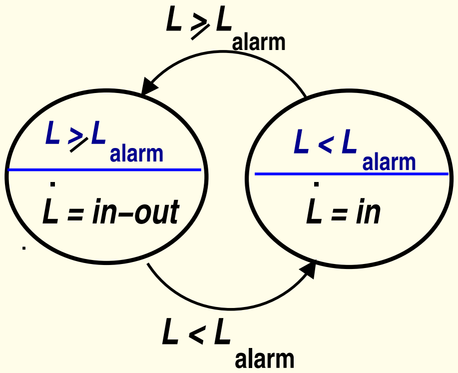
(b)(i) $\exists L, t(L_{\mathrm{alarm}} \leq L \leq L_{\mathrm{overflow}} \wedge L + \mathrm{in}(t) - \mathrm{out} > L_{\mathrm{overflow}})$ iff $\exists t(\mathrm{in}(t) > \mathrm{out})$

(ii) $\exists L, t(L < L_{\mathrm{alarm}} \wedge L + \mathrm{in}(t) > L_{\mathrm{overflow}})$ iff $\exists t(\mathrm{in}(t) > L_{\mathrm{overflow}} - L_{\mathrm{alarm}})$

(3) $L_b \leq L_{\mathrm{overflow}} \wedge \forall t(\mathrm{in}(t) \leq \mathrm{out}) \wedge \forall t(\mathrm{in}(t) \leq L_{\mathrm{overflow}} - L_{\mathrm{alarm}})$

Parametric verification for hybrid systems

Discrete control (jumps); Continuous evolution in given modes (flows).



Here special case: Linear hybrid automata

- Jump guards; updates: linear constraints between non-primed and primed variables
- Mode invariants: bounds on (linear combinations of the values of) control variables

$$\text{Inv}_q \quad \sum_{i=1}^n a_i x_i \leq a$$

- Flow conditions: boundedness conditions on (linear combinations of) slopes

$$\text{flow}_q \quad \sum_{i=1}^n c_i \dot{x}_i \leq c$$

Alternative formulation:

$$\text{Flow}_m((x_i)_{i=1,n}, 0, t) : \quad \forall t', t''(0 \leq t' \leq t'' \leq t \rightarrow \sum_{i=1}^n c_i(x_i(t'') - x_i(t')) \leq c(t'' - t'))$$

Parametric verification for hybrid systems

Discrete control (jumps); Continuous evolution in given modes (flows).

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

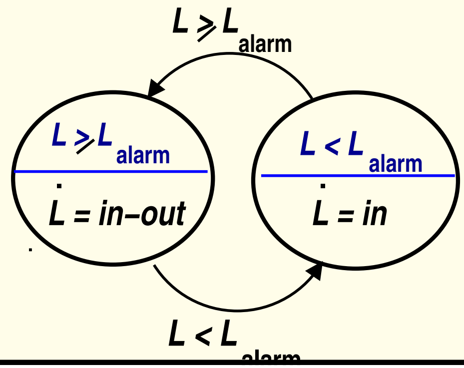
Task: Study under which conditions the following are false:

(Jump) $\exists \overline{x}, \overline{x}' (\mathsf{Inv}_m(\overline{x}) \wedge \Phi(\overline{x}) \wedge \mathsf{Jump}_{m,m'}(\overline{x}, \overline{x}') \wedge \mathsf{Inv}_{m'}(\overline{x}') \wedge \neg\Phi(\overline{x}'))$.

(Flow) $\exists t (\mathsf{Inv}_m(\overline{x}(0)) \wedge \Phi(\overline{x}(0)) \wedge \forall t' (0 \leq t' \leq t \rightarrow \mathsf{Flow}_m(\overline{x}, 0, t') \wedge \mathsf{Inv}_m(\overline{x}(t')))$
$\wedge \neg\Phi(\overline{x}(t)))$

Parametric data: Example (Water tank)

Safety condition: $L \leq L_{\text{overflow}}$



The following are equivalent

(1) The safety condition is an invariant under jumps and flows

(2) The disjunction of the following formulae is false

(a) $\exists L(L \leq L_{\text{alarm}} \wedge L \leq L_{\text{overflow}} \wedge L > L_{\text{overflow}})$ false

(b) $\exists L(L > L_{\text{alarm}} \wedge L \leq L_{\text{overflow}} \wedge L > L_{\text{overflow}})$ false

(c) $\exists L, t(L(0) < L_{\text{alarm}} \wedge \forall t'(0 \leq t' \leq t \rightarrow L(t') = L(0) + \text{in}\, t' \wedge L(t') < L_{\text{alarm}}) \wedge L(t) > L_{\text{overflow}})$ false

(d) $\exists L, t(L(0) \geq L_{\text{alarm}} \wedge \forall t'(0 \leq t' \leq t \rightarrow L(t') = L(0) + (\text{in} - \text{out})t' \wedge L(t') \geq L_{\text{alarm}})) \wedge L(t) > L_{\text{overflow}})$

iff $\text{in} - \text{out} > 0$

[Damm, Ihlemann,VS'11] PTIME algorithm for invariant checking (uses locality)
classes of LHA for which safety properties can be checked in PTIME
and bounded-time reachability is in NP

Synthesis: parametric bounds on slope $\mapsto$ constraints guaranteeing safety

General method

Verification $\mapsto$ hierarchical reasoning in local theory extensions

Synthesis $\mapsto$ hierarchical QE in local theory extensions

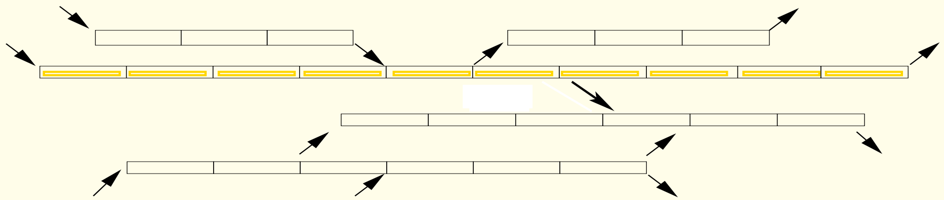
Examples:

- **Program verification:**
  - Insertion of elements in sorted arrays [VS'10]
- **Verification of controllers:**
  - Train systems [Jacobs,VS'06] [Faber,Jacobs,VS'07],[Faber,Ihlemann,Jacobs,VS'10]
  - Linear hybrid automata [Damm,Ihlemann,VS'11, VS'13]
    - A chemical plant [Damm,Ihlemann,VS'11]
    - Families of similar linear hybrid automata [VS'13] [Damm,Horbach,VS, in progress]

Example: ETCS Case Study (AVACS project)

[Faber,Ihlemann,Jacobs,VS'10]

Verification of train systems with complex track topology

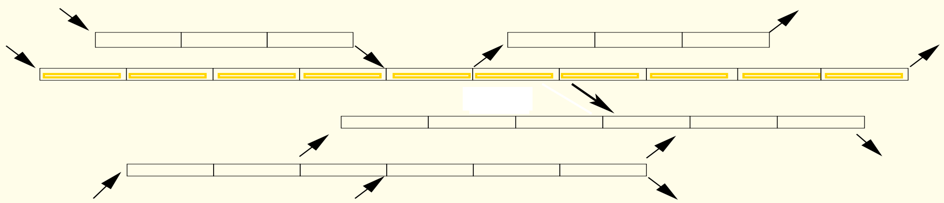


Idea: Reduce complexity by exploiting modularity at various levels
specification / verification / structurally

Main goal: exploit modularity at various levels

[Faber,Ihlemann,Jacobs,VS'10]

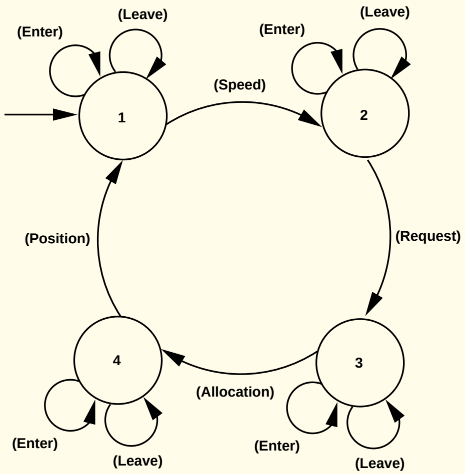
Verification of train systems with complex track topology



1. Specification

- Use the modular language COD [Hoenicke,Olderog'02], which allows us to separately specify
  - processes (as Communicating Sequential Processes, CSP),
  - data (using Object-Z, OZ), and
  - time, durations (using the Duration Calculus, DC).

Example: Controller for line track (RBC)



RBC

method *enter* : [*s*1? : *Segment*; *t*0? : *Train*; *t*1? : *Train*; *t*2? : *Train*]

method *leave* : [*ls*? : *Segment*; *lt*? : *Train*]

local_chan *alloc*, *req*, *updPos*, *updSpd*

main $\stackrel{c}{=}$ ((*enter* → main) □ (*leave* → main) □ (*updSpd* → *State*1))

*State*1 $\stackrel{c}{=}$ ((*enter* → *State*1) □ (*leave* → *State*1) □ (*req* → *State*2))

*State*2 $\stackrel{c}{=}$ ((*alloc* → *State*3) □ (*enter* → *State*2) □ (*leave* → *State*2))

*State*3 $\stackrel{c}{=}$ ((*enter* → *State*3) □ (*leave* → *State*3) □ (*updPos* → main))

CSP

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]

$sd : SegmentData$
$td : TrainData$

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

Init

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

OZ

effect_*updSpd*

$\Delta(spd)$

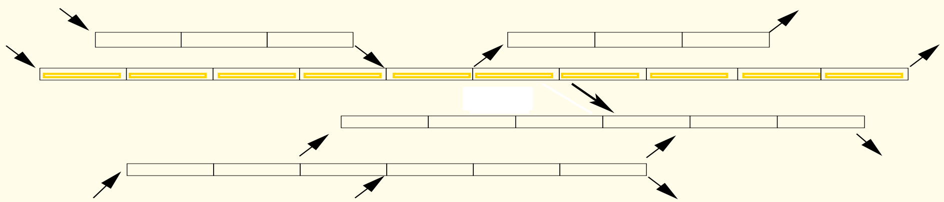
$\forall t : Train \mid pos(t) < length(segm(t)) - d \wedge spd(t) - decmax \cdot \Delta t > 0$
$\quad Γ \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq lmax(segm(t))$
$\forall t : Train \mid pos(t) \geq length(segm(t)) - d \wedge alloc(nexts(segm(t))) = tid(t)$
$\quad Γ \max\{0, spd(t) -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

⋮

Main goal: exploit modularity at various levels

[Faber,Ihlemann,Jacobs,VS'10]

Verification of train systems with complex track topology

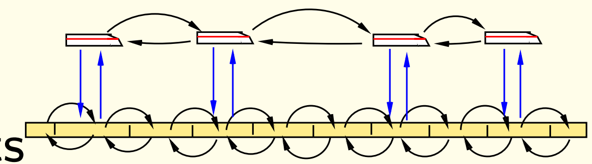


2. Verification

- **Verification tasks: linear track; incoming, outgoing trains**

  Data structures Pointers; 2 Sorts: Trains

  Segments



$\mapsto$ Safety checking: reasoning in complex data structures

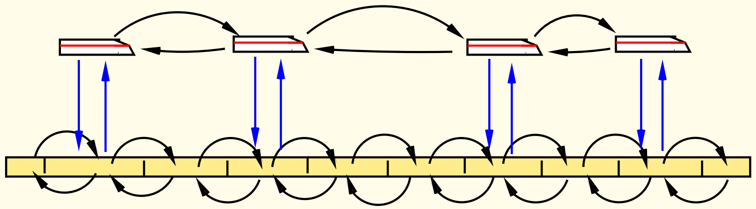
$\mapsto$ Solution: hierarchical and modular reasoning

Modular Verification

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

Modular Verification

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



Example 1: Speed Update

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

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

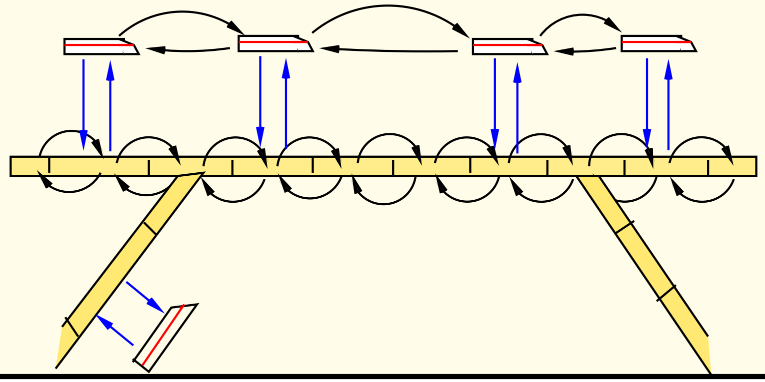
$$\text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \;\wedge\; \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t) \;\rightarrow\; \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0)$$

Proof task:

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

Modular Verification

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



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

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

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

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

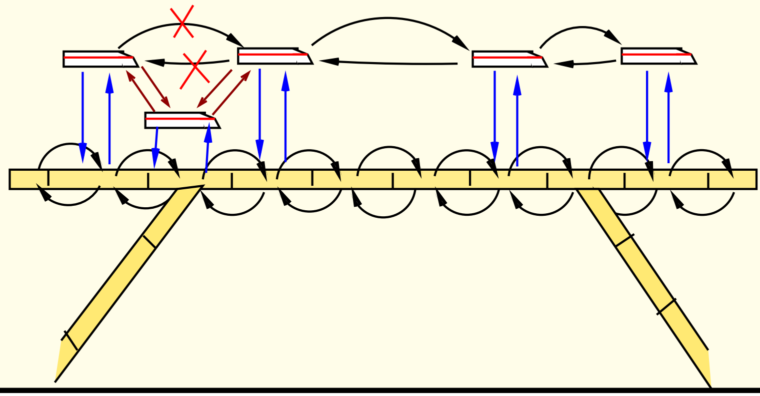
$\rightarrow \text{next}'(t){=}\text{next}_t(t)$

...

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

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($\overline{x}, \overline{x}'$)



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

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

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

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

...

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

Safety property

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

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

Our solution

Find an 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 .

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

Modularity in automated reasoning

Problem: Axioms, Invariants, Updates: are universally quantified

Examples of theories we need to handle

- **Invariants**

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

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

Modularity in automated reasoning

Problem: Axioms, Invariants, Updates: are universally quantified

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

Problem: Axioms, Invariants, Updates: are universally quantified

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**

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

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

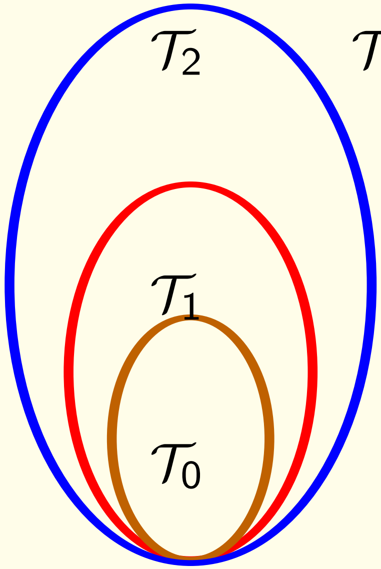
The good news

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:

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



$UIF \cup \mathbb{R}$

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

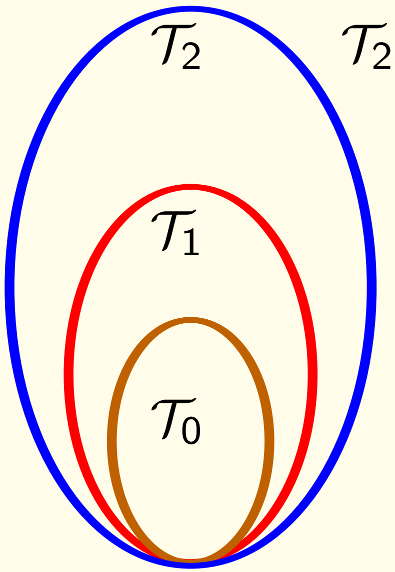
$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Inv}(\mathsf{next}, \dots)$$

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

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



$UIF \cup \mathbb{R}$

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

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

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

To show:

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

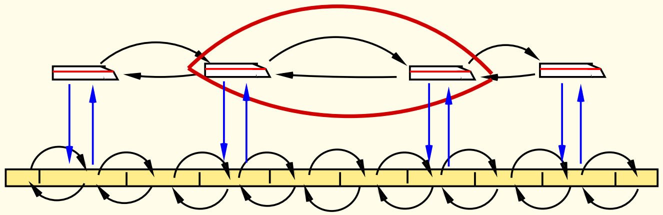
$$UIF \cup \mathbb{R} \cup (\mathsf{PointerAx}[G''] \cup G'')_0 \models \perp$$

H-PILoT: verification/ QE $\mapsto$ constr. on param.

model building/counterexample generation

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($\overline{x}, \overline{x}'$)



Example 1: Speed Update

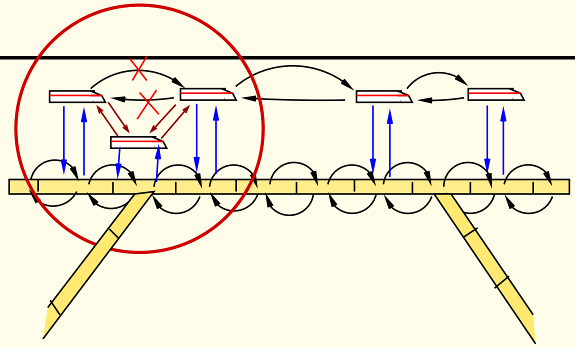
$$\begin{array}{lcl} \text{pos}(t) < \text{length}(\text{segm}(t)) - d & \rightarrow & 0 \leq \text{spd}'(t) \leq \text{lmax}(\text{segm}(t)) \\ \text{pos}(t) \geq \text{length}(\text{segm}(t)) - d & \wedge & \text{alloc}(\text{next}_s(\text{segm}(t))) = \text{tid}(t) \\ & \rightarrow & 0 \leq \text{spd}'(t)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

Proof task:

Inv(pos, next, prev, spd) $\wedge$ SpeedUpdate(pos, next, prev, spd, spd$'$) $\rightarrow$ Inv(pos$'$, next, prev, spd$'$)

Modular Verification

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



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

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

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

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

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

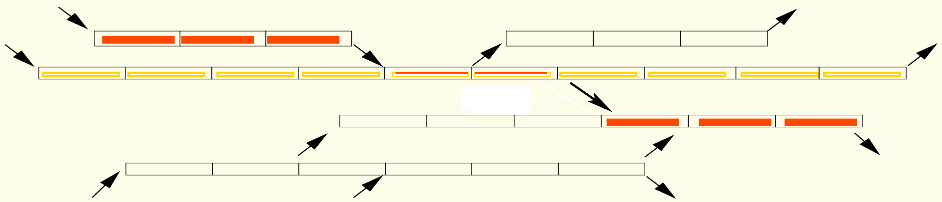
...

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

Main goal: exploit modularity at various levels

[Faber,Ihlemann,Jacobs,VS'10]

Verification of train systems with complex track topology



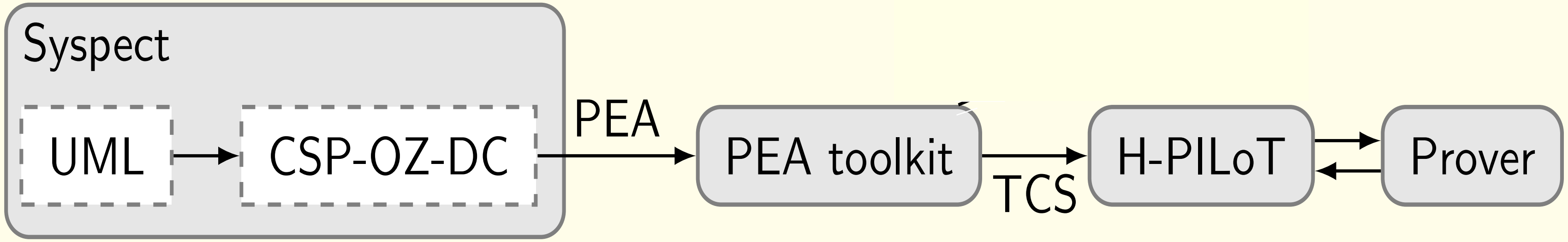
3. Structurally

↦ Complex track topology (Assumption: No cycles; degree at most 2)

- decomposition into family of linear tracks (may overlap)
- prove that safety of whole system follows from
  (1) safety for the controller of a linear track and
  (2) compatibility of controllers on jointly controlled trains.

• Synthesis: - Constraints on parameters which guarantee safety

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)

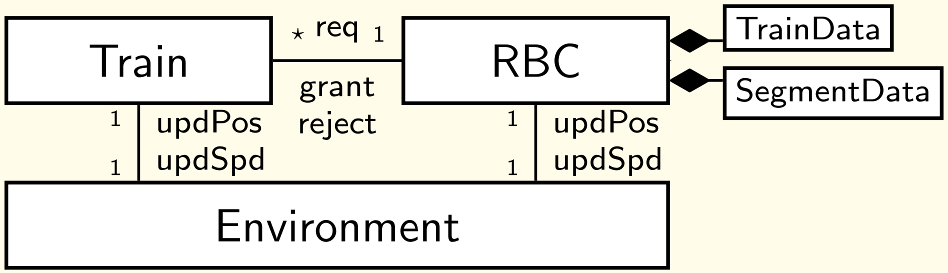
Main advantage: Capability of H-PILoT of detecting satisfiability and constructing counterexamples $\mapsto$ correct specifications.

Timed train controller (Train)

Note: The correctness proof for the whole system proceeds as follows:

(1) We proved safety of the RBC under the assumption that the trains have certain properties.

(2) We prove that the trains indeed satisfy such properties (or determine conditions on parameters under which such properties hold).



CSP-OZ-DC specification for **Train**.

(1) Verification

(2) Synthesis of constraints for which **Train** satisfies the safety requirements.

Conclusion

- **Local theory extensions**

  Limit search / modularity: hierarchic reasoning

  Recognize locality: embeddability; saturation

  Combine various extensions:
  $\mapsto$ Modular reasoning; information exchange

- **Applications**

  Here: Verification

Summary

Theory	Applications
Efficient reasoning	**Verification** (AVACS)
• Theories	• **Deductive verification** case studies with Damm, Faber, Ihlemann, Jacobs
• Theory extensions	
• Chains of theory extensions	• **Synthesis** first steps
• Theory combinations	• **Abstraction refinement**
Hierarchic, modular reasoning	with A. Rybalchenko
Parameterized complexity	• **Model generation** $\mapsto$ planning?
Model generation	**Cryptography** first steps
Implementation H-PILoT	**Knowledge representation**
Interpolation	with F. Gasse
Complex systems	Verification: Modularity; case studies

References

- **A. Armando, M.P. Bonacina, S. Ranise, and St. Schulz.**
  On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal.
  Proceedings of FroCos'05, LNCS 3717, pages 65-80. Springer Verlag, 2005.

- **A. Armando, S. Ranise, and M. Rusinowitch.**
  A rewriting approach to satisfiability procedures.
  Information and Computation, 183(2):140-164, 2003.

- **D. Basin and H. Ganzinger.**
  Automated complexity analysis based on ordered resolution.
  Journal of the ACM, 48(1):70-109, 2001.

- **D.A. Basin and H. Ganzinger.**
  Complexity analysis based on ordered resolution.
  Proceedings of LICS'96, pages 456-465.
  IEEE Computer Society Press, 1996.

- **W. Damm, C. Ihlemann, V. Sofronie-Stokkermans.**
  Decidability and complexity for the verification of safety properties of reasonable linear hybrid automata.
  Proceedings of HSCC 2011, pages 73-82, ACM.

- **W. Damm, C. Ihlemann, V. Sofronie-Stokkermans.**
  PTIME Parametric Verification of Safety Properties for Reasonable Linear Hybrid Automata.
  Mathematics in Computer Science 5(4): 469-497 (2011)

References (ctd.)

- **J. Faber, C. Ihlemann, S. Jacobs, V. Sofronie-Stokkermans.**
  Automatic Verification of Parametric Specifications with Complex Topologies.
  Proceedings of IFM 2010, LNCS 6396, pages 152-167, Springer.

- **J. Faber, S. Jacobs, and V. Sofronie-Stokkermans.**
  Verifying CSP-OZ-DC specifications with complex data types and timing parameters.
  Proceedings of IFM 2007, LNCS 4591, pages 233-252. Springer, 2007.

- **H. Ganzinger.**
  Relating semantic and proof-theoretic concepts for polynomial time decidability of uniform word problems.
  Proceedings of 16th IEEE Symposium on Logic in Computer Science (LICS'01), pages 81-92. IEEE Computer Society Press, 2001.

- **H. Ganzinger, V. Sofronie-Stokkermans, and U. Waldmann.**
  Modular proof systems for partial functions with Evans equality.
  Information and Computation, 204(10):1453-1492, 2006.
  (extended version of a paper which appeared in the proceedings of IJCAR'04)

- **S. Ghilardi.**
  Model theoretic methods in combined constraint satisfiability.
  Journal of Automated Reasoning, 33(3-4):221-249, 2004.

- **R. Givan and D. McAllester.**
  New results on local inference relations.
  In Principles of Knowledge Representation and reasoning: Proceedings of KR'92, pages 403-412. Morgan Kaufmann Press, 1992.

References (ctd.)

- **R. Givan and D.A. McAllester.**
  Polynomial-time computation via local inference relations.
  ACM Transactions on Computational Logic, 3(4):521-541, 2002.

- **J. Hoenicke and E.-R. Olderog.**
  CSP-OZ-DC: A combination of specification techniques for processes, data and time.
  Nordic Journal of Computing, 9(4):301-334, 2002. Appeared March 2003.

- **M. Horbach, V. Sofronie-Stokkermans.**
  Obtaining Finite Local Theory Axiomatizations via Saturation.
  Proceedings of FroCos 2013, LNCS 8152, pages 198-213, Springer.

- **C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans.**
  On local reasoning in verification.
  Proceedings of TACAS 2008, LNCS 4963, pages 265-281, Springer 2008.

- **C. Ihlemann, V. Sofronie-Stokkermans.**
  System Description: H-PILoT. Proceedings of CADE 2009, LNCS 5663, pages 131-139, Springer.

- **C. Ihlemann, V. Sofronie-Stokkermans.**
  On Hierarchical Reasoning in Combinations of Theories. Proceedings of IJCAR 2010, LNCS 6173, pages 30-45.

- **S. Jacobs and V. Sofronie-Stokkermans.**
  Applications of hierarchical reasoning in the verification of complex systems.
  Electronic Notes in Theoretical Computer Science, 174(8):39-54, 2007.

References (ctd.)

- **S. McPeak and G.C. Necula.**
  Data structure specifications via local equality axioms.
  Proceedings of CAV 2005, LNCS 3576, pages 476-490, 2005.

- **G. Nelson and D.C. Oppen.**
  Simplification by cooperating decision procedures.
  ACM Transactions on Programming Languages and Systems, 1979.

- **A. Rybalchenko and V. Sofronie-Stokkermans.**
  Constraint solving for interpolation.
  Proceedings of VMCAI 2007, LNCS 4349, pages 346-362. Springer, 2007.

  An extended version appeared in J. Symb. Comput. 45(11): 1212-1233 (2010).

- **V. Sofronie-Stokkermans.**
  Hierarchic reasoning in local theory extensions.
  Proceedings of CADE-20, LNAI 3632, pages 219-234, 2005. Springer.

- **V. Sofronie-Stokkermans.**
  Interpolation in local theory extensions.
  Proceedings of IJCAR 2006, LNAI 4130, pages 235-250. Springer, 2006.

  An extended version appeared in Logical Methods in Computer Science 4(4) (2008)

- **V. Sofronie-Stokkermans.**
  Hierarchical and Modular Reasoning in Complex Theories: The Case of Local Theory Extensions.
  Proceedings of FroCoS 2007, LNCS 4720, pages 47-71, Springer.

References (ctd.)

- **V. Sofronie-Stokkermans.**
  Efficient Hierarchical Reasoning about Functions over Numerical Domains.
  Proceedings of KI 2008. LNAI 5243, pages 135-143, Springer 2008.

- **V. Sofronie-Stokkermans.**
  Locality Results for Certain Extensions of Theories with Bridging Functions.
  Proceedings of CADE 2009, LNCS 5663, pages 67-83, Springer.

- **V. Sofronie-Stokkermans.**
  Hierarchical Reasoning for the Verification of Parametric Systems.
  Proceedings of IJCAR 2010, LNCS 6173, pages 171-187, Springer.

- **V. Sofronie-Stokkermans.**
  Hierarchical Reasoning and Model Generation for the Verification of Parametric Hybrid Systems.
  Proceedings of CADE 2013, LNCS 7898, pages 360-376, Springer.

- **C. Tinelli and C. Ringeissen.**
  Unions of non-disjoint theories and combinations of satisfiability procedures.
  Theoretical Computer Science, 290(1):291-353, 2003.

- **C. Tinelli and C. Zarba.**
  Combining nonstably infinite theories.
  Journal of Automated Reasoning, 34(3):209-238, 2005.

- . . .