Decision Procedures for Verification

Combinations of decision procedures (4)

10.02.2015

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Until Now

- Decision procedures for specific theories

  in some cases for satisfiability of ground formulae only

- Combinations of decision procedures

  Nelson/Oppen (for conjunctions of ground literals)

- DPLL(T)

  for conjunctions of ground clauses

- Formulae with quantifiers

  a theory of arrays (started last time)

A theory of arrays

We consider the theory of arrays in a many-sorted setting.

Syntax:

- Sorts: Elem (elements), Array (arrays) and Index (indices, here integers).
- Function symbols: read, write.

$$a(\text{read}) = \mathit{Array} \times \mathit{Index} \rightarrow \mathit{Element}$$

$$a(\text{write}) = \mathit{Array} \times \mathit{Index} \times \mathit{Element} \rightarrow \mathit{Array}$$

Theories of arrays

We consider the theory of arrays in a many-sorted setting.

Theory of arrays $\mathcal{T}_{arrays}$:

- $\mathcal{T}_i$ (theory of indices): Presburger arithmetic
- $\mathcal{T}_e$ (theory of elements): arbitrary
- Axioms for read, write

$$\begin{array}{rcl} read(write(a,i,e),i) & \approx & e \\ j \not\approx i \vee read(write(a,i,e),j) & = & read(a,j). \end{array}$$

Theories of arrays

We consider the theory of arrays in a many-sorted setting.

Theory of arrays $\mathcal{T}_{arrays}$:

- $\mathcal{T}_i$ (theory of indices): Presburger arithmetic
- $\mathcal{T}_e$ (theory of elements): arbitrary
- Axioms for read, write

$$\begin{array}{rcl} read(write(a, i, e), i) & \approx & e \\ j \not\approx i \vee read(write(a, i, e), j) & = & read(a, j). \end{array}$$

Fact: Undecidable in general.

Goal: Identify a fragment of the theory of arrays which is decidable.

A decidable fragment

- **Index guard** a positive Boolean combination of atoms of the form $t \leq u$ or $t = u$ where t and u are either a variable or a ground term of sort Index

  Example: $(x \leq 3 \vee x \approx y) \wedge y \leq z$ is an index guard

  Example: $x + 1 \leq c, \quad x + 3 \leq y, \quad x + x \leq 2$ are not index guards.

- **Array property formula** [Bradley,Manna,Sipma'06]

  $(\forall i)(\varphi_I(i) \rightarrow \varphi_V(i))$, where:

  φ_I: index guard

  φ_V: formula in which any universally quantified i occurs in a direct array read; no nestings

  Example: $c \leq x \leq y \leq d \rightarrow a(x) \leq a(y)$ is an array property formula

  Example: $x < y \rightarrow a(x) < a(y)$ is not an array property formula

Decision Procedure

(Rules should be read from top to bottom)

Step 1: Put F in NNF.

Step 2: Apply the following rule exhaustively to remove writes:

$$\frac{F[\mathit{write}(a, i, v)]}{F[a'] \wedge a'[i] = v \wedge (\forall j. j \neq i \rightarrow a[j] = a'[j])} \quad \text{for fresh } a' \text{ (write)}$$

Given a formula F containing an occurrence of a write term $\mathit{write}(a, i, v)$, we can substitute every occurrence of $\mathit{write}(a, i, v)$ with a fresh variable a' and explain the relationship between a' and a.

Decision Procedure

Step 3 Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists i.\, G[i]]}{F[G[j]]} \text{ for fresh } j \text{ (exists)}$$

Existential quantification can arise during Step 1 if the given formula contains a negated array property.

Decision Procedure

Steps 4-6 accomplish the reduction of universal quantification to finite conjunction.

The main idea is to select a set of symbolic index terms on which to instantiate all universal quantifiers.

Theories of arrays

Step 4 From the output F3 of Step 3, construct the index set $\mathcal{I}$:

$$\begin{aligned}\mathcal{I} \quad = \quad & \{\lambda\}\cup \\ & \{t \mid \cdot[t] \in F3 \text{ such that } t \text{ is not a universally quantified variable}\}\cup \\ & \{t \mid t \text{ occurs as an } \textit{evar} \text{ in the parsing of index guards}\}\end{aligned}$$

(evar is any constant, ground term, or unquantified variable.)

This index set is the finite set of indices that need to be examined. It includes all terms t that occur in some $\mathit{read}(a, t)$ anywhere in F (unless it is a universally quantified variable) and all terms t that are compared to a universally quantified variable in some index guard.

λ is a fresh constant that represents all other index positions that are not explicitly in $\mathcal{I}$.

Theories of arrays

Step 5 Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\forall \bar{i}. F[i] \rightarrow G[i]]}{H\left[\bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}])\right]} \quad \text{(forall)}$$

where n is the size of the list of quantified variables $\bar{i}$.

This is the key step.

It replaces universal quantification with finite conjunction over the index set. The notation $\bar{i} \in \mathcal{I}^n$ means that the variables $\bar{i}$ range over all n-tuples of terms in $\mathcal{I}$.

Theories of arrays

Step 6: From the output $F5$ of Step 5, construct

$$F6: \quad F5 \wedge \bigwedge_{i \in \mathcal{I} \setminus \{\lambda\}} \lambda \neq i$$

The new conjuncts assert that the variable λ introduced in Step 4 is unique: it does not equal any other index mentioned in F5.

Step 7: Decide the TA-satisfiability of $F6$ using the decision procedure for the quantifier free fragment.

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i])$$

It contains one array property,

$$\forall i . i \neq l \rightarrow a[i] = b[i]$$

index guard: $i \neq l := (i \leq l - 1 \vee i \geq l + 1)$ value constraint: $a[i] = b[i]$

Step 1: The formula is already in NNF.

Step 2: We rewrite F as:

$$\begin{aligned} F2 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j . j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i])$$

Step 2: We rewrite F as:

$$\begin{aligned} F2: \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j . j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

index guards: $i \neq l := (i \leq l - 1 \vee i \geq l + 1)$ value constraint: $a[i] = b[i]$

$i \neq l := (j \leq l - 1 \vee j \geq l + 1)$ value constraint: $a[i] = a'[j]$

Step 3: F2 does not contain any existential quantifiers $\mapsto$ F3 = F2.

Step 4: The index set is

$$\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l, l-1, l+1\} = \{\lambda, k, l, l-1, l+1\}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i])$$

Step 3:

$$\begin{aligned} F3 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j. j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Step 4: $\mathcal{I} = \{\lambda, k, l, l - 1, l + 1\}$

Step 5: we replace universal quantification as follows:

$$\begin{aligned} F5 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \bigwedge_{i \in \mathcal{I}} (i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge \bigwedge_{i \in \mathcal{I}} (j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\forall i . i \neq l \rightarrow a[i] = b[i])$$

$$\mathcal{I} = \{\lambda, k, l, l-1, l+1\}$$

Step 5 (continued) Expanding produces:

$$\begin{aligned}
F5' : \quad & a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land \\
& (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \land (k \neq l \rightarrow a[k] = b[k]) \land (l \neq l \rightarrow a[l] = b[l]) \land \\
& (l-1 \neq l \rightarrow a[l-1] = b[l-1]) \land (l+1 \neq l \rightarrow a[l+1] = b[l+1]) \land \\
& a'[l] = v \land (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \land (k \neq l \rightarrow a[k] = a'[k]) \land \\
& (l \neq l \rightarrow a[l] = a'[l]) \land (l-1 \neq l \rightarrow a[l-1] = a'[l-1]) \land \\
& (l+1 \neq l \rightarrow a[l+1] = a'[l+1]).
\end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i])$$

$$\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l\} = \{\lambda, k, l\}$$

Step 5 (continued): Simplifying produces

$$\begin{aligned}
F''5 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \\
& \wedge (k \neq l \rightarrow a[k] = b[k]) \wedge a[l-1] = b[l-1] \wedge a[l+1] = b[l+1] \\
& \wedge a'[l] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \\
& \wedge (k \neq l \rightarrow a[k] = a'[k]) \wedge a[l-1] = a'[l-1] \wedge a[l+1] = a'[l+1].
\end{aligned}$$

Example

Consider the array property formula

$$F : \mathit{write}(a, l, v)[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\forall i.\, i \neq l \rightarrow a[i] = b[i])$$

Step 6 distinguishes λ from other members of I:

$$\begin{aligned}
F6 : \quad & a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \\
& \land (k \neq l \rightarrow a[k] = b[k]) \land a[l-1] = b[l-1] \land a[l+1] = b[l+1] \\
& \land a'[l] = v \land (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \\
& \land (k \neq l \rightarrow a[k] = a'[k]) \land a[l-1] = a'[l-1] \land a[l+1] = a'[l+1] \\
& \land \lambda \neq k \land \lambda \neq l \land \lambda \neq l-1 \land \lambda \neq l+1.
\end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i.i \neq l \rightarrow a[i] = b[i])$$

Step 6 Simplifying, we have

$$\begin{aligned} F'6 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge a[\lambda] = b[\lambda] \\ & \wedge a[k] = b[k] \wedge a[l-1] = b[l-1] \wedge a[l+1] = b[l+1] \\ & \wedge a'[l] = v \wedge a[\lambda] = a'[\lambda] \\ & \wedge (k \neq l \rightarrow a[k] = a'[k]) \wedge a[l-1] = a'[l-1] \wedge a[l+1] = a'[l+1] \\ & \wedge \lambda \neq k \wedge \lambda \neq l \wedge \lambda \neq l-1 \wedge \lambda \neq l+1. \end{aligned}$$

We can use for instance DPLL(T).
Alternative: Case distinction. There are two cases to consider.

(1) If $k{=}l$, then $a'[l]{=}v$ and $a'[k]{=}b[k]$ imply $b[k]{=}v$, yet $b[k]{\neq}v$.
(2) If $k{\neq}l$, then $a[k]{=}v$ and $a[k]{=}b[k]$ imply $b[k]{=}v$, but again $b[k]{\neq}v$.

Hence, F'6 is TA-unsatisfiable, indicating that F is TA-unsatisfiable.

Soundness and Completeness

Theorem (Soundness and Completeness)

Consider a formula F from the array property fragment . The output F6 of Step 6 is T_{arrays}-equisatisfiable to F.

Proof

(Soundness) Step 1-6 preserve satisfiability

(Fi is a logical consequence of F$i-1$).

Soundness and Completeness

Theorem (Soundness and Completeness)

Consider a formula F from the array property fragment . The output F6 of Step 6 is T_{arrays}-equisatisfiable to F.

Proof (Completeness)

Step 6: From the output $F5$ of Step 5, construct

$$F6: \quad F5 \wedge \bigwedge_{i \in \mathcal{I} \setminus \{\lambda\}} \lambda \neq i$$

Assume that $F6$ is satisfiabile. Clearly F5 has a model.

Soundness and Completeness

Theorem (Soundness and Completeness)

Consider a formula F from the array property fragment . The output F6 of Step 6 is T_{arrays}-equisatisfiable to F.

Proof (Completeness)

Step 5 Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\forall \bar{i}. F[i] \rightarrow G[i]]}{H\left[\bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}])\right]} \quad \text{(forall)}$$

Assume that $F5$ is satisfiabile. Let $\mathcal{A} = (\mathbb{Z}, \text{Elem}, \{a_A\}_{a \in Arrays}, \ldots)$ be a model for F5. Construct a model $\mathcal{B}$ for $F4$ as follows.

For $x \in \mathbb{Z}$: $l(x)$ $(u(x))$ closest left (right) neighbor of x in $\mathcal{I}$.

$$a_{\mathcal{B}}(x) = \begin{cases} a_{\mathcal{A}}(l(x)) & \textit{if } x - l(x) \leq u(x) - x \text{ or } u(x) = \infty \\ a_{\mathcal{A}}(u(x)) & \textit{if } x - l(x) > u(x) - x \text{ or } l(x) = -\infty \end{cases}$$

Soundness and Completeness

Theorem (Soundness and Completeness)

Consider a formula F from the array property fragment . The output F6 of Step 6 is T_{arrays}-equisatisfiable to F.

Proof (Completeness)

Step 3 Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists i. G[i]]}{F[G[j]]} \text{ for fresh } j \text{ (exists)}$$

If F3 has model then F2 has model

Soundness and Completeness

Theorem (Soundness and Completeness)

Consider a formula F from the array property fragment . The output F6 of Step 6 is T_{arrays}-equisatisfiable to F.

Proof (Completeness)

Step 2: Apply the following rule exhaustively to remove writes:

$$\frac{F[write(a, i, v)]}{F[a'] \wedge a'[i] = v \wedge (\forall j. j \neq i \rightarrow a[j] = a'[j])} \quad \text{for fresh } a' \text{ (write)}$$

Given a formula F containing an occurrence of a write term $write(a, i, v)$, we can substitute every occurrence of $write(a, i, v)$ with a fresh variable a' and explan the relationship between a' and a.

If F2 has a model then F1 has a model.

Step 1: Put F in NNF: NNF F1 is equivalent to F.

Theories of arrays

Theorem (Complexity) Suppose $(T_{index} \cup T_{elem})$-satisfiability is in NP. For sub-fragments of the array property fragment in which formulae have bounded-size blocks of quantifiers, T_{arrays}-satisfiability is NP-complete.

Proof NP-hardness is clear.

That the problem is in NP follows easily from the procedure: instantiating a block of n universal quantifiers quantifying subformula G over index set I produces $|I| \cdot n$ new subformulae, each of length polynomial in the length of G. Hence, the output of Step 6 is of length only a polynomial factor greater than the input to the procedure for fixed n.

Program verification

Example: Does BubbleSort return a sorted array?

```
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}
```

Program Verification

$-1 \leq i < |a| \wedge$
$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$
$\text{sorted}(a, i, |a|-1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$
$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$
$\text{sorted}(a, i, |a|-1)$
$\text{partitioned}(a, 0, j-1, j, j)$ C_2

Example: Does BubbleSort return a sorted array?

```
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}
```

Generate verification conditions and prove that they are valid

Predicates:

- sorted(a, l, u): $\forall i, j (l \leq i \leq j \leq u \rightarrow a[i] \leq a[j])$
- partitioned(a, l_1, u_1, l_2, u_2): $\forall i, j (l_1 \leq i \leq u_1 \leq l_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j])$

Program Verification

Example: Does BubbleSort return a sorted array?

```
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}
```

$-1 \leq i < |a| \wedge$
$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$
$\text{sorted}(a, i, |a|-1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$
$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$
$\text{sorted}(a, i, |a|-1)$
$\text{partitioned}(a, 0, j-1, j, j)$ $\quad C_2$

Generate verification conditions and prove that they are valid

Predicates:

- $\text{sorted}(a, l, u)$: $\quad \forall i, j (l \leq i \leq j \leq u \rightarrow a[i] \leq a[j])$
- $\text{partitioned}(a, l_1, u_1, l_2, u_2)$: $\quad \forall i, j (l_1 \leq i \leq u_1 \leq l_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j])$

To prove: $C_2(a) \wedge \text{Update}(a, a') \rightarrow C_2(a')$

Another Situation

Insertion of an element c in a sorted array a of length n

$$
\begin{array}{l}
\text{for } (i := 1; i \leq n; i := i + 1)\ \{ \\
\qquad\qquad \text{if } a[i] \geq c\{n := n + 1 \\
\qquad\qquad\qquad\qquad\qquad \text{for } (j := n; j > i; j := j - 1)\{a[i] := a[i - 1]\} \\
\qquad\qquad\qquad\qquad\qquad a[i] := c;\ \text{return } a \\
\}\}\ a[n + 1] := c;\ \text{return } a
\end{array}
$$

Task:

If the array was sorted before insertion it is sorted also after insertion.

$\text{Sorted}(a, n) \wedge \text{Update}(a, n, a', n') \wedge \neg\text{Sorted}(a', n') \models_{\mathcal{T}} \perp$?

Another Situation

Task:

If the array was sorted before insertion it is sorted also after insertion.

$\mathsf{Sorted}(a, n) \wedge \mathsf{Update}(a, n, a', n') \wedge \neg\mathsf{Sorted}(a', n') \models_{\mathcal{T}} \perp$?

$\mathsf{Sorted}(a, n)$	$\forall i, j(1 \leq i \leq j \leq n \rightarrow a[i] \leq a[j])$
$\mathsf{Update}(a, n, a', n')$	$\forall i((1 \leq i \leq n \wedge a[i] < c) \rightarrow a'[i] = a[i])$
	$\forall i((c \leq a(1) \rightarrow a'[1] := c)$
	$\forall i((a[n] < c \rightarrow a'[n+1] := c)$
	$\forall i((1 \leq i-1 \leq i \leq n \wedge a[i-1] < c \wedge a[i] \geq c) \rightarrow (a'[i] = c)$
	$\forall i((1 \leq i-1 \leq i \leq n \wedge a[i-1] \geq c \wedge a[i] \geq c \rightarrow a'[i] := a[i-1])$
	$n' := n+1$
$\neg\mathsf{Sorted}(a', n')$	$\exists k, l(1 \leq k \leq l \leq n' \wedge a'k] > a'[l])$

Beyond the array property fragment

Extension: New arrays defined by case distinction – $\mathrm{Def}(f')$

$$\forall \overline{x}(\phi_i(\overline{x}) \rightarrow f'(\overline{x}){=}s_i(\overline{x})) \qquad i \in I, \text{ where } \phi_i(\overline{x}) \wedge \phi_j(\overline{x}) \models_{\mathcal{T}_0} \perp \text{ for } i{\neq}j \, (1)$$

$$\forall \overline{x}(\phi_i(\overline{x}) \rightarrow t_i(\overline{x}){\leq}f'(\overline{x}){\leq}s_i(\overline{x})) \qquad i \in I, \text{ where } \phi_i(\overline{x}) \wedge \phi_j(\overline{x}) \models_{\mathcal{T}_0} \perp \text{ for } i{\neq}j \, (2)$$

where s_i, t_i are terms over the signature Σ such that $\mathcal{T}_0 \models \forall \overline{x}(\phi_i(\overline{x}){\rightarrow}t_i(\overline{x}){\leq}s_i(\overline{x}))$ for all $i \in I$.

$\mathcal{T}_0 \subseteq \mathcal{T}_0 \wedge \mathrm{Def}(f')$ has the property that for every set G of ground clauses in which there are no nested applications of f':

$$\mathcal{T}_0 \wedge \mathrm{Def}(f') \wedge G \models \perp \quad \text{iff} \quad \mathcal{T}_0 \wedge \mathrm{Def}(f')[G] \wedge G$$

(sufficient to use instances of axioms in $\mathrm{Def}(f')$ which are relevant for G)

• Some of the syntactic restrictions of the array property fragment can be lifted

Example

Insertion in an array

(on the blackboard)

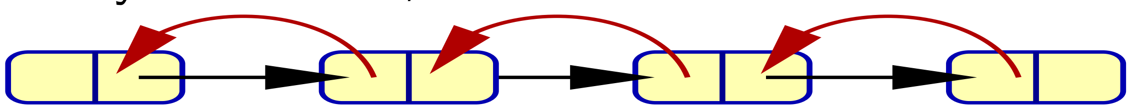
Pointer Structures

[McPeak, Necula 2005]

- pointer sort p, scalar sort s; pointer fields (p $\rightarrow$ p); scalar fields (p $\rightarrow$ s);
- axioms: $\forall p \;\; \mathcal{E} \vee \mathcal{C}$; $\mathcal{E}$ contains disjunctions of pointer equalities
  $\mathcal{C}$ contains scalar constraints

Assumption: If $f_1(f_2(\ldots f_n(p)))$ occurs in axiom, the axiom also contains:
$$p{=}\text{null} \vee f_n(p){=}\text{null} \vee \cdots \vee f_2(\ldots f_n(p))){=}\text{null}$$

Example: doubly-linked lists; ordered elements



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

Pointer Structures

[McPeak, Necula 2005]

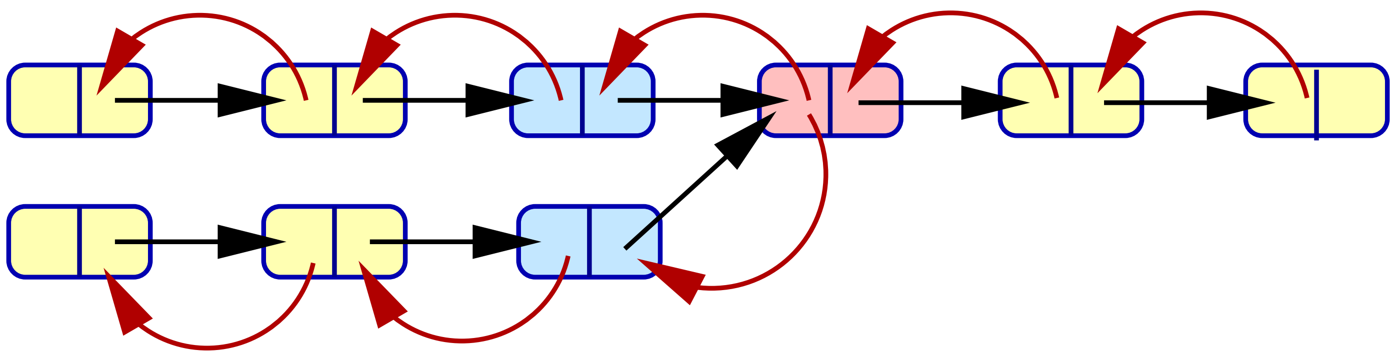
- pointer sort p, scalar sort s; pointer fields (p $\rightarrow$ p); scalar fields (p $\rightarrow$ s);
- axioms: $\forall p \;\; \mathcal{E} \vee \mathcal{C}$; $\quad$ $\mathcal{E}$ contains disjunctions of pointer equalities
  $\mathcal{C}$ contains scalar constraints

Assumption: If $f_1(f_2(\dots f_n(p)))$ occurs in axiom, the axiom also contains:
$$p{=}\text{null} \vee f_n(p){=}\text{null} \vee \dots \vee f_2(\dots f_n(p))){=}\text{null}$$

Theorem. K set of clauses in the fragment above. Then for every set G of ground clauses, $(K \cup G) \cup \mathcal{T}_s \models \perp$ iff $K^{[G]} \cup \mathcal{T}_s \models \perp$

where $K^{[G]}$ is the set of instances of K in which the variables are replaced by subterms in G.

Example: A theory of doubly-linked lists

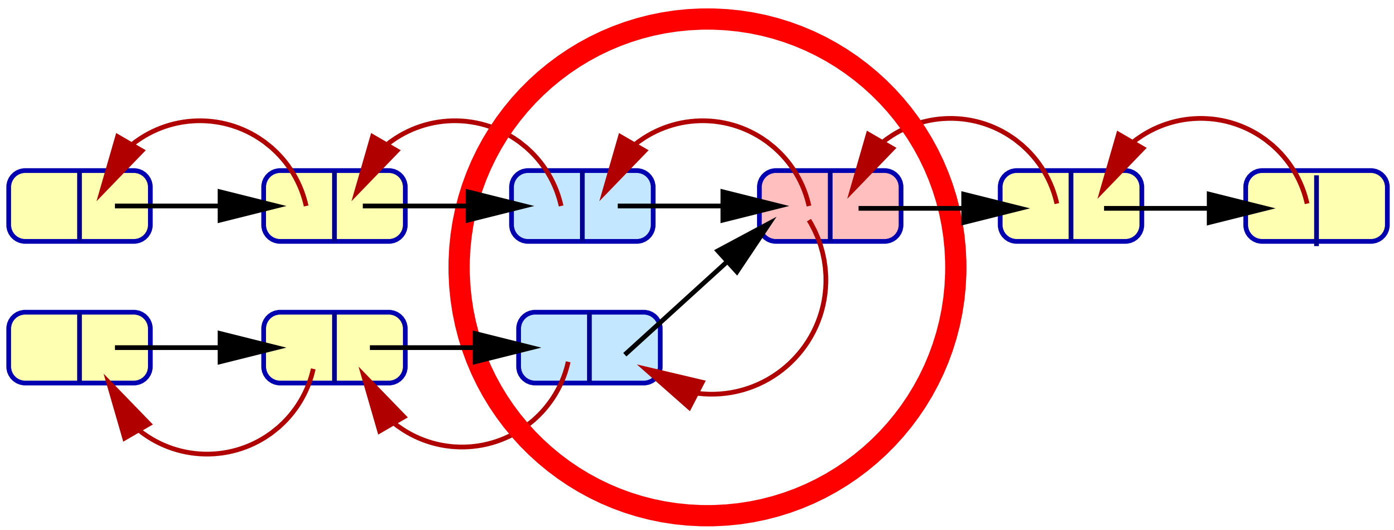


$$\forall p\ (p \neq \text{null} \land p.\text{next} \neq \text{null} \ \rightarrow\ p.\text{next}.\text{prev} = p)$$

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

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

Example: A theory of doubly-linked lists

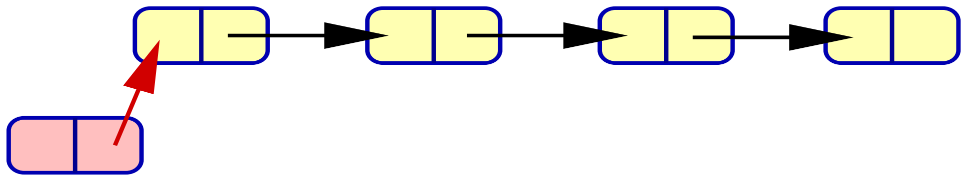


$(c \neq \mathrm{null} \wedge c.\mathrm{next} \neq \mathrm{null} \rightarrow c.\mathrm{next}.\mathrm{prev} = c)$ $\quad$ $(c.\mathrm{next} \neq \mathrm{null} \wedge c.\mathrm{next}.\mathrm{next} \neq \mathrm{null} \rightarrow c.\mathrm{next}.\mathrm{next}.\mathrm{prev} = c.\mathrm{nex}$

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

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

Example: List insertion



Initially list is sorted: $p.\mathrm{next} \neq \mathrm{null} \rightarrow p.\mathrm{prio} \geq p.\mathrm{next.prio}$

$c.\mathrm{prio} = x,\ c.\mathrm{next} = \mathrm{null}$

for all $p \neq c$ **do**

if $p.\mathrm{prio} \leq x$ **then if** $\mathrm{First}(p)$ **then** $c.\mathrm{next}' = p,\ \mathrm{First}'(c),\ \neg\mathrm{First}'(p)$ **endif**; $p.\mathrm{next}' = p.\mathrm{next}$

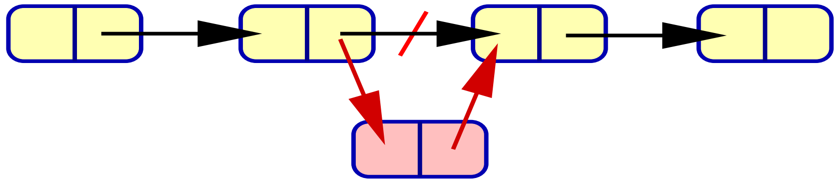
$p.\mathrm{prio} > x$ **then** **case** $p.\mathrm{next} = \mathrm{null}$ **then** $p.\mathrm{next}' := c,\ c.\mathrm{next}' = \mathrm{null}$

$p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next.prio} > x$ **then** $p.\mathrm{next}' = p.\mathrm{next}$

$p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next.prio} \leq x$ **then** $p.\mathrm{next}' = c,\ c.\mathrm{next}' = p.\mathrm{nex}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $p.\mathrm{next} \neq \mathrm{null} \rightarrow p.\mathrm{prio} \geq p.\mathrm{next}.\mathrm{prio}$

$c.\mathrm{prio} = x,\ c.\mathrm{next} = \mathrm{null}$

for all $p \neq c$ **do**

if $p.\mathrm{prio} \leq x$ **then if** $\mathrm{First}(p)$ **then** $c.\mathrm{next}' = p,\ \mathrm{First}'(c),\ \neg\mathrm{First}'(p)$ **endif**; $p.\mathrm{next}' = p.\mathrm{next}$

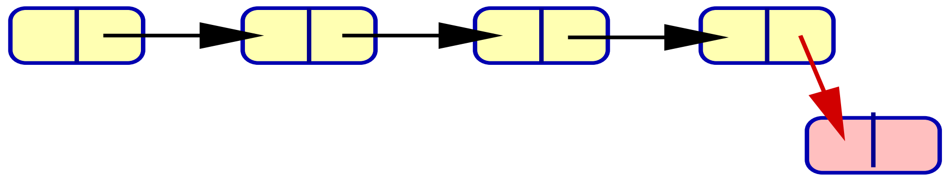
$p.\mathrm{prio} > x$ **then** **case** $p.\mathrm{next} = \mathrm{null}$ **then** $p.\mathrm{next}' := c,\ c.\mathrm{next}' = \mathrm{null}$

$p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{prio} > x$ **then** $p.\mathrm{next}' = p.\mathrm{next}$

$p.\mathrm{next} \neq \mathrm{null} \wedge p.\mathrm{next}.\mathrm{prio} \leq x$ **then** $p.\mathrm{next}' = c,\ c.\mathrm{next}' = p.\mathrm{next}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next}.\text{prio}$

$c.\text{prio} = x, \; c.\text{next} = \text{null}$

for all $p \neq c$ **do**

if $p.\text{prio} \leq x$ **then if** $\text{First}(p)$ **then** $c.\text{next}' = p, \; \text{First}'(c), \; \neg\text{First}'(p)$ **endif**; $p.\text{next}' = p.\text{next}$

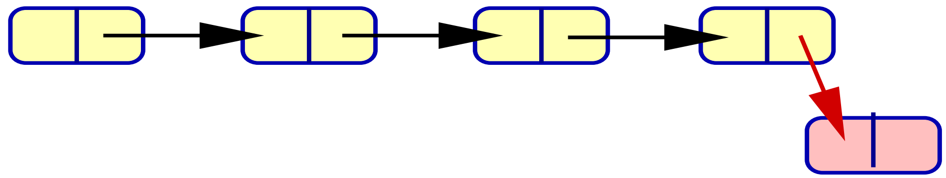
$p.\text{prio} > x$ **then** **case** $p.\text{next} = \text{null}$ **then** $p.\text{next}' := c, \; c.\text{next}' = \text{null}$

$p.\text{next} \neq \text{null} \wedge p.\text{next}.\text{prio} > x$ **then** $p.\text{next}' = p.\text{next}$

$p.\text{next} \neq \text{null} \wedge p.\text{next}.\text{prio} \leq x$ **then** $p.\text{next}' = c, \; c.\text{next}' = p.\text{nex}$

Verification task: After insertion list remains sorted

Example: List insertion



Initially list is sorted: $\forall p(p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next}.\text{prio})$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \text{First}(p) \rightarrow \text{next}'(c){=}p \wedge \text{First}'(c))$
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \text{First}(p) \rightarrow \text{next}'(p){=}\text{next}(p) \wedge \neg\text{First}'(p))$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){\leq}x \wedge \neg\text{First}(p) \rightarrow \text{next}'(p){=}\text{next}(p))$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){=}\text{null} \rightarrow \text{next}'(p){=}c$
$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){=}\text{null} \rightarrow \text{next}'(c){=}\text{null})$

$\forall p(p{\neq}\text{null} \wedge p{\neq}c \wedge \text{prio}(p){>}x \wedge \text{next}(p){\neq}\text{null} \wedge \text{prio}(\text{next}(p)){>}x \rightarrow \text{next}'(p){=}\text{next}(p))$

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

We only need to use instances in which variables are replaced by ground subterms occurring in the problem

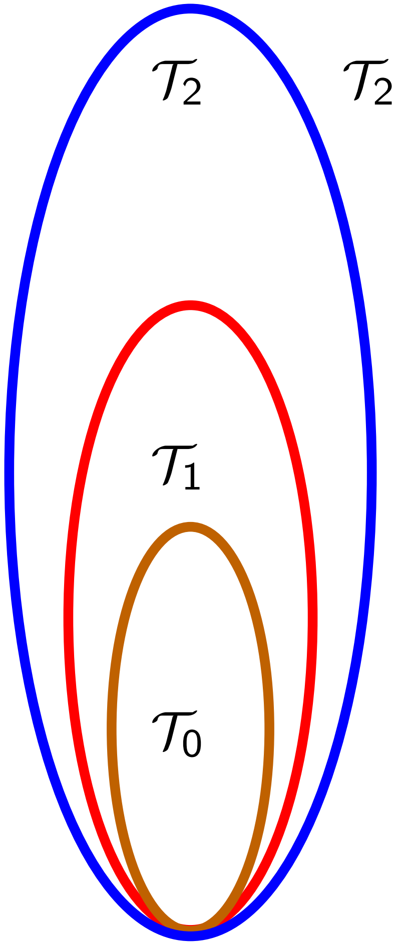
---

To check: $\text{Sorted}(\text{next}, \text{prio}) \wedge \text{Update}(\text{next}, \text{next}') \wedge p_0.\text{next}'{\neq}\text{null}{\wedge}p_0.\text{prio}{\not\geq}p_0.\text{next}'.\text{prio} \models \perp$

Example: List insertion

To show:

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

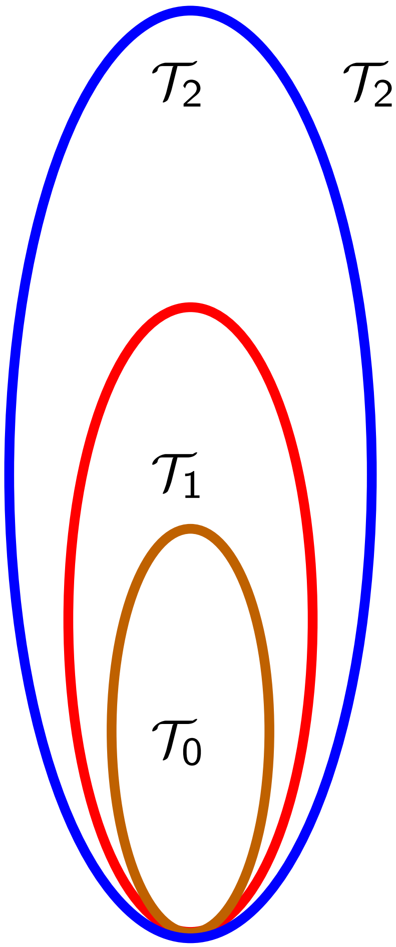


$$\mathcal{T}_2 = \mathcal{T}_1 \cup \mathsf{Update}(\mathsf{next}, \mathsf{next'})$$

$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Sorted}(\mathsf{next})$$

$$\mathcal{T}_0 = (\mathsf{Lists}, \mathsf{next})$$

Example: List insertion



To show:

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

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

Instantiate: $\mathcal{T}_1 \cup \underbrace{\boxed{\mathsf{Update}(\mathsf{next}, \mathsf{next}')[G]} \cup G}_{G'} \models$

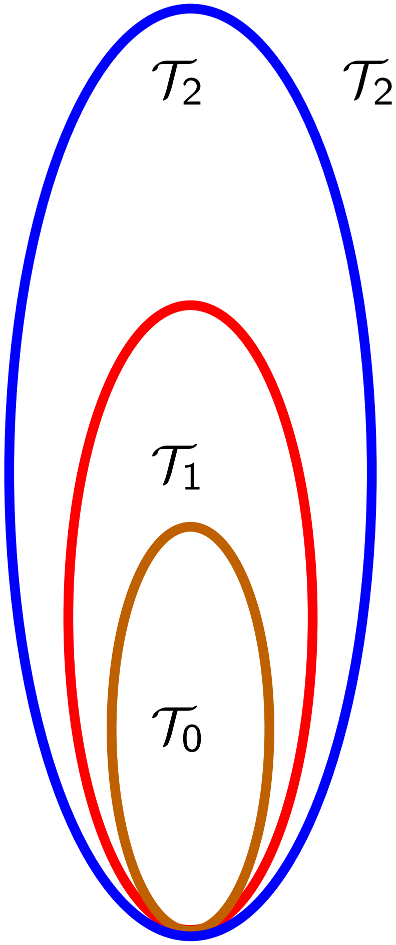
Hierarchical reasoning:

$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Sorted}(\mathsf{next})$

$\mathcal{T}_1 \cup G'(\mathsf{next}) \models \perp$

$\mathcal{T}_0 = (\mathsf{Lists}, \mathsf{next})$

Example: List insertion



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

$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Sorted}(\mathsf{next})$

$\mathcal{T}_0 = (\mathsf{Lists}, \mathsf{next})$

To show:

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

$$\Downarrow$$

$$\mathcal{T}_1 \cup G'(\mathsf{next}) \models \perp$$

$$\Downarrow$$

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

More general concept

Local Theory Extensions

Satisfiability of formulae with quantifiers

Goal: generalize the ideas for extensions of theories

Example: Strict monotonicity

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

$$\mathsf{Mon}(f) \qquad \forall i, j (i < j \rightarrow f(i) < f(j))$$

Problems:

- A prover for $\mathbb{R} \cup \mathbb{Z}$ does not know about f
- A prover for first-order logic may have problems with the reals and integers
- DPLL(T) cannot be used (Mon, $\mathbb{Z}, \mathbb{R}$: non-disjoint signatures)
- SMT provers may have problems with the universal quantifiers

Our goal: reduce search: consider certain instances Mon$(f)[G]$
without loss of completeness

hierarchical/modular reasoning:
reduce to checking satisfiability of a set of constraints over $\mathbb{R} \cup \mathbb{Z}$

Local theory extensions

Solution: Local theory extensions

$\mathcal{K}$ set of equational clauses; $\mathcal{T}_0$ theory; $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$

(Loc) $\mathcal{T}_0 \subseteq \mathcal{T}_1$ is local, if for ground clauses G,
$\mathcal{T}_0 \cup \mathcal{K} \cup G \models \perp$ iff $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ has no (partial) model

Various notions of locality, depending of the instances to be considered:
stable locality, order locality; extended locality.

Example: Strict monotonicity

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

Base theory ($\mathbb{R} \cup \mathbb{Z}$)	Extension
$a < b$	$f(a) = f(b) + 1$
	$\forall i, j (i < j \rightarrow f(i) < f(j))$

Example: Strict monotonicity

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

Extension is local $\mapsto$ replace axiom with ground instances

Base theory ($\mathbb{R} \cup \mathbb{Z}$)	Extension
$a < b$	$f(a) = f(b) + 1$
	$a < b \rightarrow f(a) < f(b)$
	$b < a \rightarrow f(b) < f(a)$

Solution 1:

$SMT(\mathbb{R} \cup \mathbb{Z} \cup UIF)$

Example: Strict monotonicity

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

Extension is local $\mapsto$ replace axiom with ground instances

Add congruence axioms. Replace pos-terms with new constants

Base theory ($\mathbb{R} \cup \mathbb{Z}$)	Extension
$a < b$	$f(a) = f(b) + 1$
	$a < b \rightarrow f(a) < f(b)$
	$b < a \rightarrow f(b) < f(a)$
	$a = b \rightarrow f(a) = f(b)$

Solution 2:

Hierarchical reasoning

Example: Strict monotonicity

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

Extension is local $\mapsto$ replace axiom with ground instances

Replace f-terms with new constants

Add definitions for the new constants

Base theory ($\mathbb{R} \cup \mathbb{Z}$)	Extension
$a < b$	$a_1 = b_1 + 1$
	$a < b \rightarrow a_1 < b_1$
	$b < a \rightarrow b_1 < a_1$
	$a = b \rightarrow a_1 = b_1$

Example: Strict monotonicity

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

Extension is local $\mapsto$ replace axiom with ground instances

Replace f-terms with new constants

Add definitions for the new constants

Base theory ($\mathbb{R} \cup \mathbb{Z}$)	Extension
$a < b$	$a_1 = f(a)$
$a_1 = b_1 + 1$	$b_1 = f(b)$
$a < b \rightarrow a_1 < b_1$	
$b < a \rightarrow b_1 < a_1$	
$a = b \rightarrow a_1 = b_1$	

Reasoning in local theory extensions

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

Problem: Decide whether $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G \models \perp$

Solution 1: Use $SMT(\mathcal{T}_0 + UIF)$: possible only if $\mathcal{K}[G]$ ground

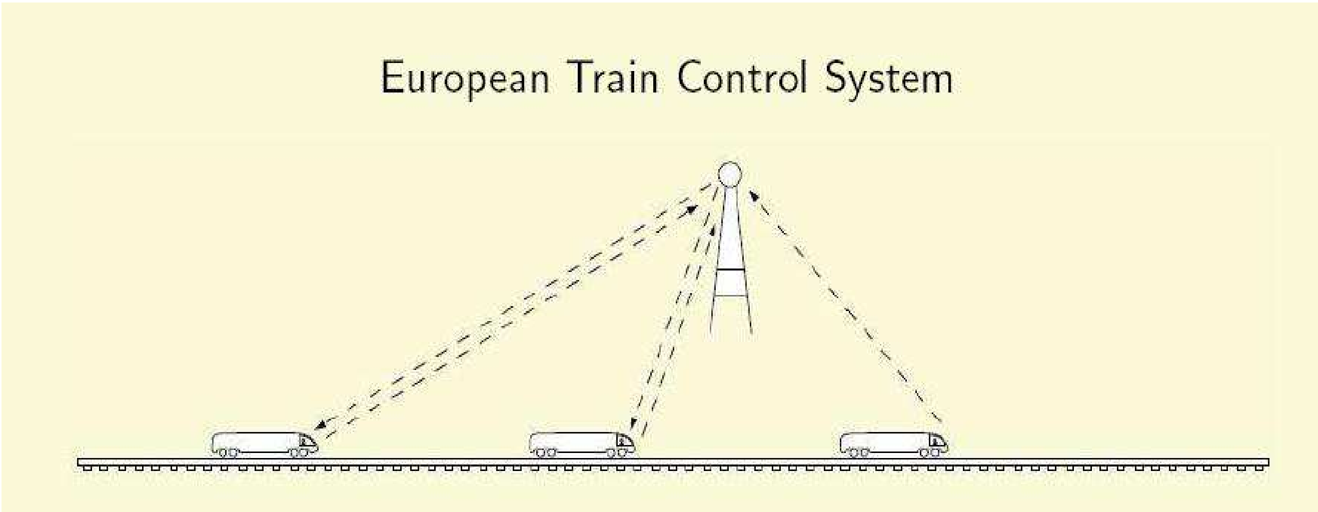
Solution 2: Hierarchic reasoning [VS'05]

reduce to satisfiability in $\mathcal{T}_0$: applicable in general

$\mapsto$ parameterized complexity

Example

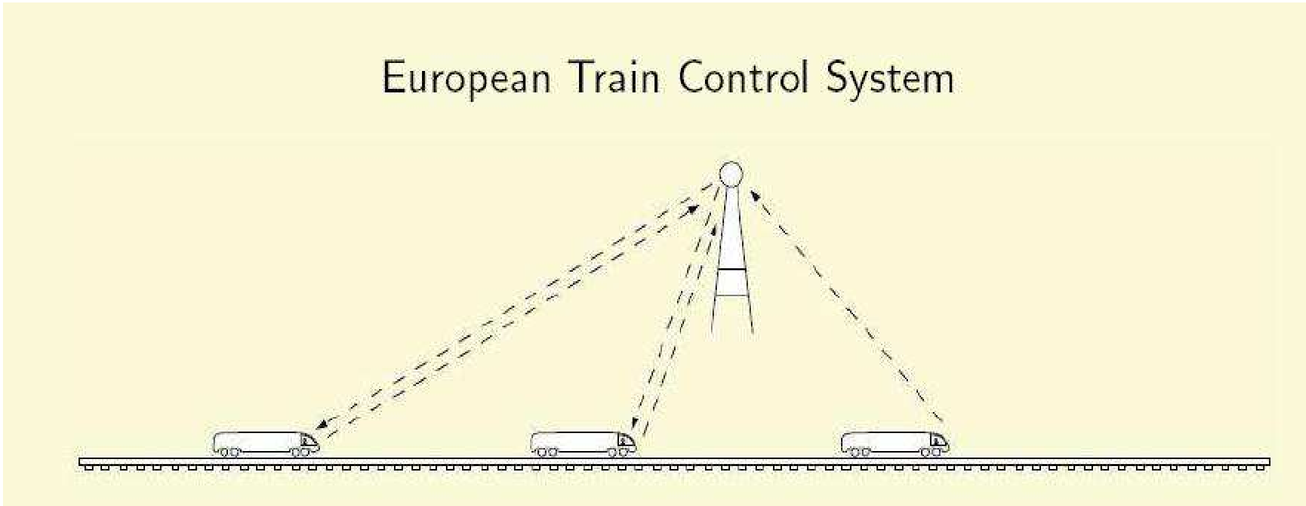
Simplified version of ETCS Case Study [Jacobs,VS'06, Faber,Jacobs,VS'07]



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

Example

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



Update(pos, pos′) :

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

...

Example

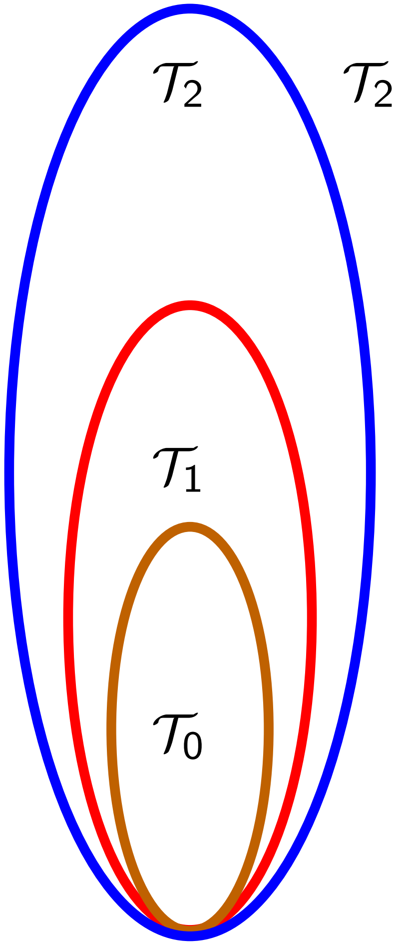
Safety property: No collisions $\quad$ $\text{Safe}(\text{pos}): \quad \forall i, j (i < j \rightarrow \text{pos}(i) > \text{pos}(j))$

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

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

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

Example



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

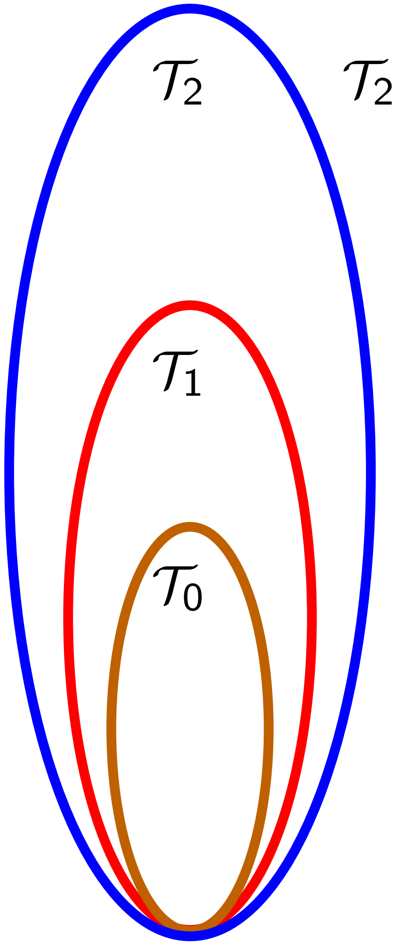
$$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Safe}(\mathsf{pos})$$

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

To show:

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

Example



To show:

$$\mathcal{T}_2 = \mathcal{T}_1 \cup \text{Update}(\text{pos}, \text{pos}') \qquad \mathcal{T}_2 \cup \underbrace{\neg\text{Safe}(\text{pos}')}_{G} \models \perp$$

$$\Downarrow$$

$$\mathcal{T}_1 = \mathcal{T}_0 \cup \text{Safe}(\text{pos}) \qquad \mathcal{T}_1 \cup G'(\text{pos}) \models \perp$$

$$\Downarrow$$

$$\mathcal{T}_0 = \mathbb{R} \cup \mathbb{Z} \qquad \mathcal{T}_0 \cup G'' \models \perp$$

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

Method 1: SAT checking/ Counterexample generation

Method 2: Quantifier elimination

relationships between parameters which guarantee safety

More complex ETCS Case studies

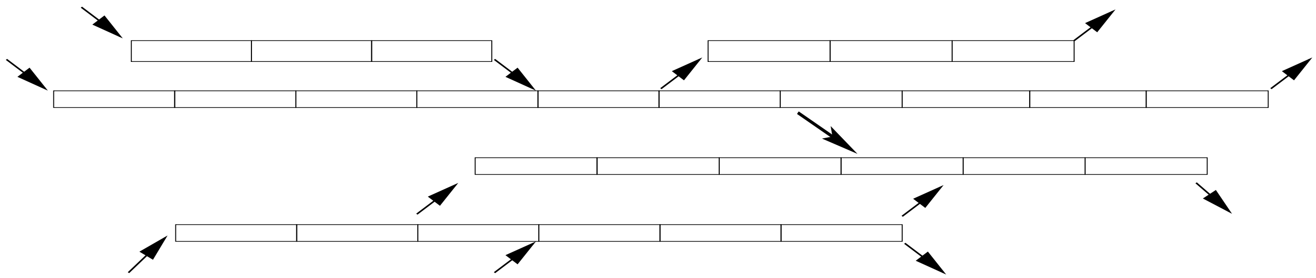
[Faber, Jacobs, VS, 2007]

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

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

Example 2: Parametric topology

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

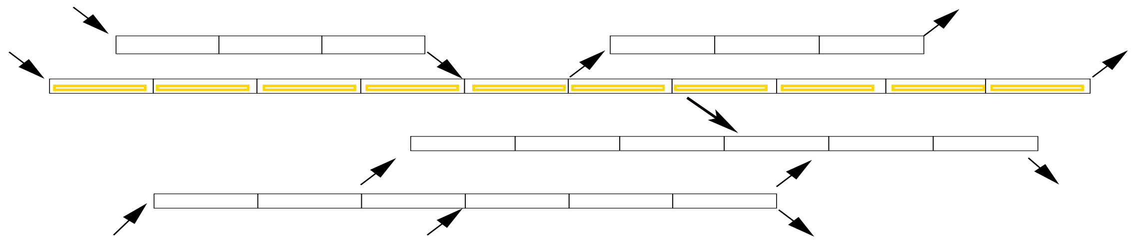


Assumptions:

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

Parametricity and modularity

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



Assumptions:

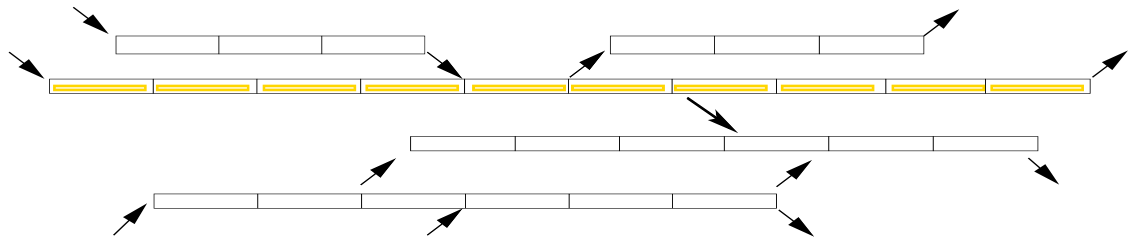
- No cycles
- in-degree (out-degree) of associated graph at most 2.

Approach:

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

Parametricity and modularity

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

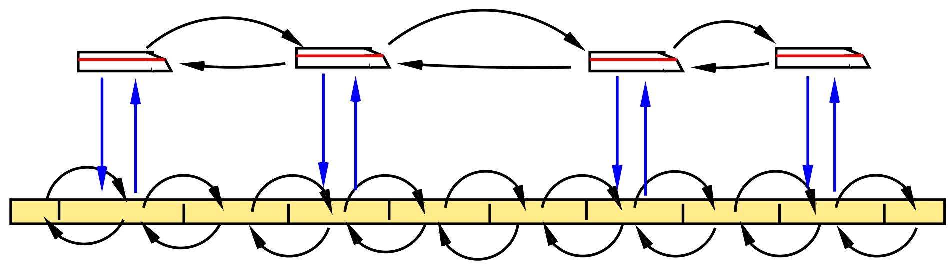


Assumptions:

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

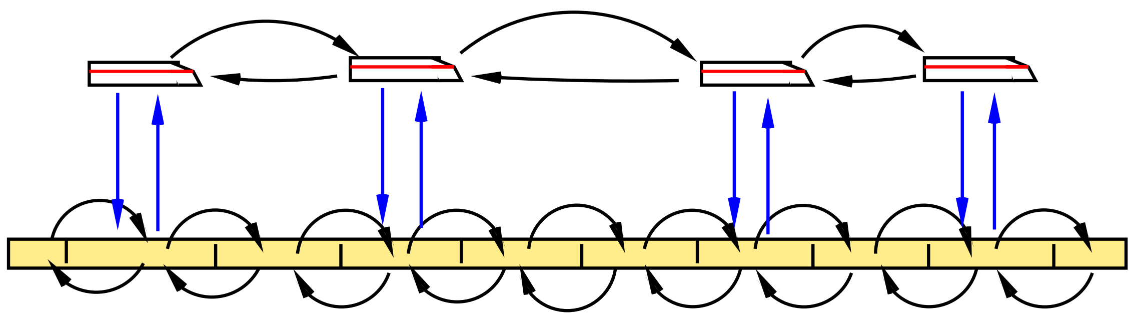
Data structures:

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



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

Incoming and outgoing trains



Example 1: Speed Update

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

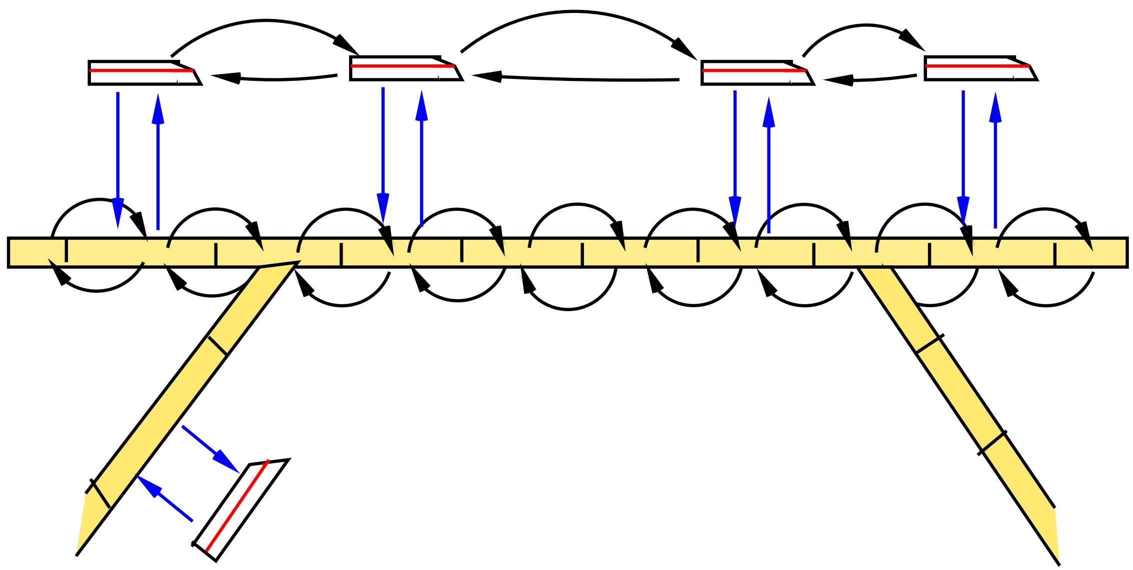
$$\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d \;\wedge\; \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) = \mathsf{tid}(t)$$

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

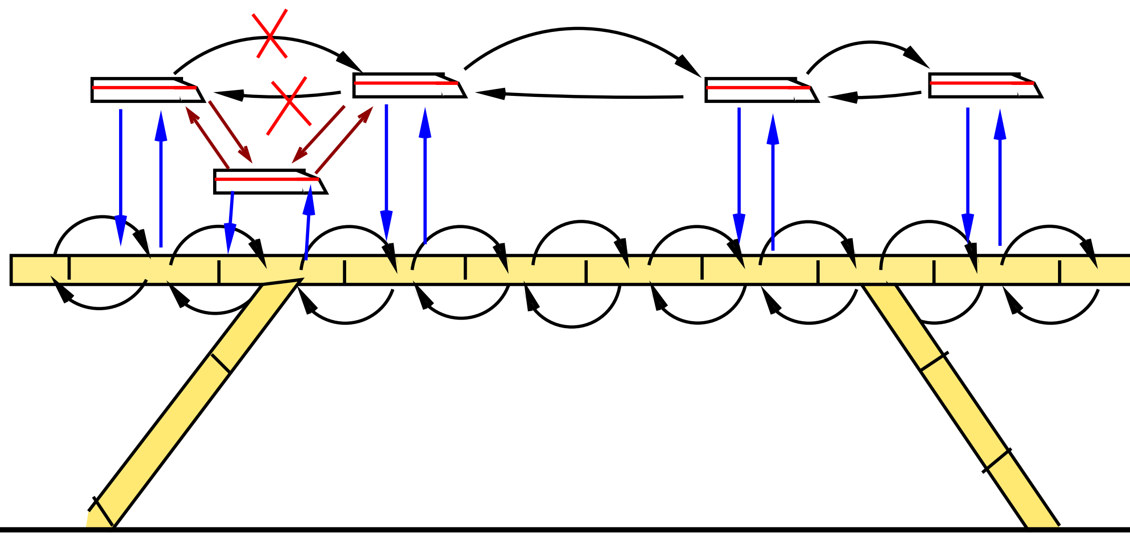
$$\mathsf{pos}(t) \geq \mathsf{length}(\mathsf{segm}(t)) - d \;\wedge\; \mathsf{alloc}(\mathsf{next}_s(\mathsf{segm}(t))) \neq \mathsf{tid}(t)$$

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

Incoming and outgoing trains



Incoming and outgoing trains



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

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

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

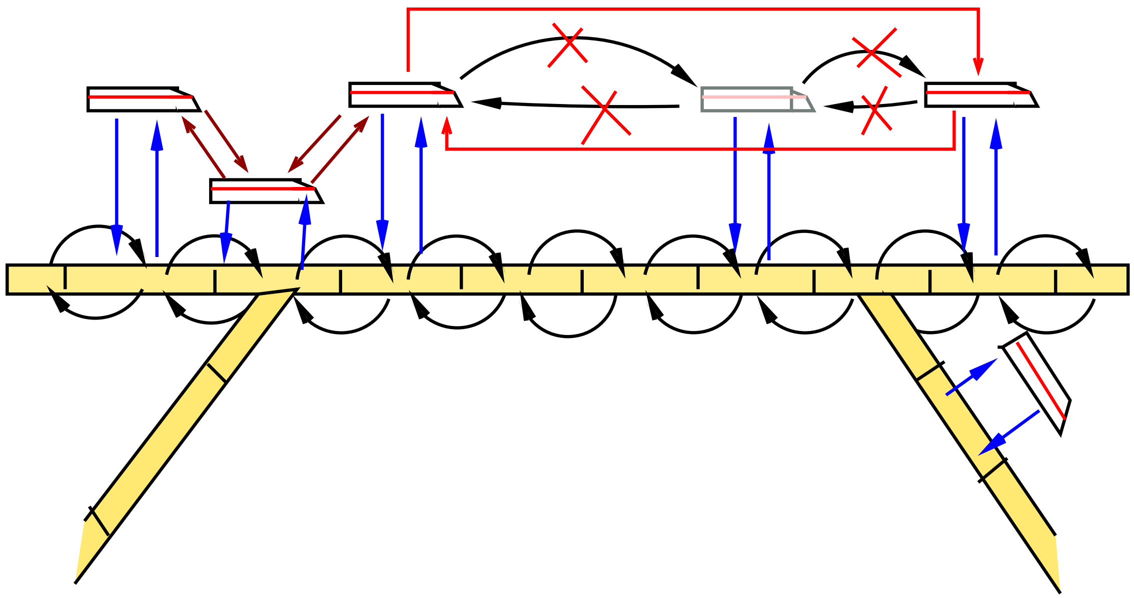
$t{\neq}t_1, \text{ids}(\text{segm}(t)){<}\text{ids}(s_1), \text{alloc}(s_1){=}\text{tid}(t_1), \text{next}_t(t){\neq}\text{null}_t, \text{ids}(\text{segm}(\text{next}_t(t))){\leq}\text{ids}(s_1)$

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

...

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

Incoming and outgoing trains



Safety property

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

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

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

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

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

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

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

Safety property

Need additional invariants.

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

  use the capabilities of H-PILoT of generating counterexamples

- generate automatically [work in progress]

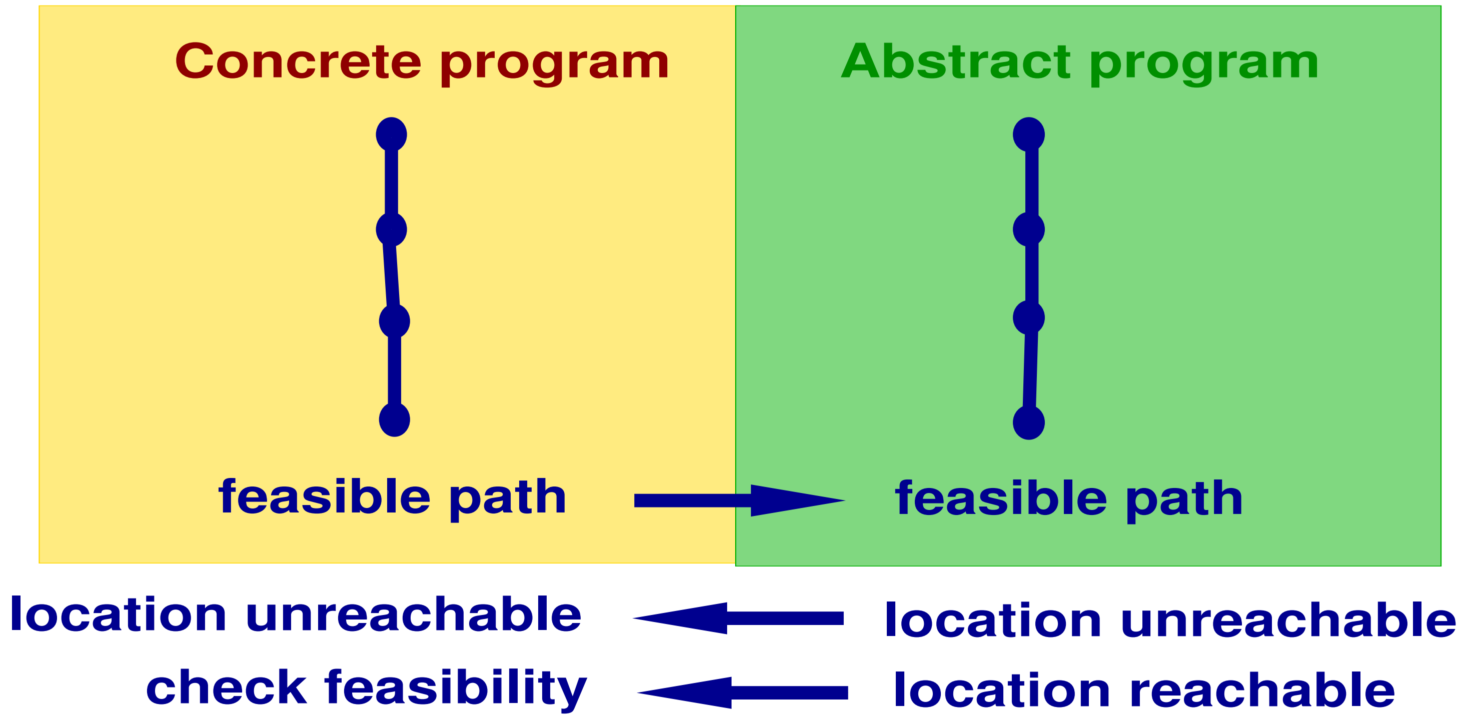
Ground satisfiability problems for pointer data structures

the decision procedures presented before can be used without problems

Other interesting topics

- Generate invariants
- Verification by abstraction/refinement

Abstraction-based Verification



$\Downarrow$

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

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

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

$\mapsto$ compute interpolants

Summary

- Decision procedures for various theories/theory combinations

  Implemented in most of the existing SMT provers:
  Z3: `http://z3.codeplex.com/`
  CVC4: `http://cvc4.cs.nyu.edu/web/`
  Yices: `http://yices.csl.sri.com/`

- Ideas about how to use them for verification

Decision procedures for other classes of theories/Applications"

Next semester: Seminar "Decision Procedures and Applications"

More details on Specification, Model Checking, Verification:

This summer (end of August):

Summer school "Verification Technology, Systems & Applications"

Next year: Lecture "Formal Specification and Verification"

Forschungspraktika

BSc/MSc Theses in the area