Decision Procedures for Verification

Combinations of Decision Procedures (3)

7.02.2022

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de
Until now

Combinations of Decision Procedures

The Nelson/Oppen Procedure

(for theories with disjoint signature)

From conjunctions to arbitrary combinations

DPLL(T)
Satisfiability of formulae with quantifiers
Satisfiability of formulae with quantifiers

In many applications we are interested in testing the satisfiability of formulae containing (universally quantified) variables.

Examples

• check satisfiability of formulae in the Bernays-Schönfinkel class

• check whether a set of (universally quantified) Horn clauses entails a ground clause

• check whether a property is consequence of a set of axioms

Example 1: $f : \mathbb{Z} \rightarrow \mathbb{Z}$ is monotonely increasing and $g : \mathbb{Z} \rightarrow \mathbb{Z}$ is defined by $g(x) = f(x + x)$ then $g$ is also monotonely increasing

Example 2: If array $a$ is increasingly sorted, and $x$ is inserted before the first position $i$ with $a[i] > x$ then the array remains increasingly sorted.
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.

\[
\begin{align*}
    a(\text{read}) &= Array \times \text{Index} \to Element \\
    a(\text{write}) &= Array \times \text{Index} \times Element \to Array
\end{align*}
\]
Theories of arrays

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

**Theory of arrays** $T_{\text{arrays}}$:

- $T_i$ (theory of indices): Presburger arithmetic
- $T_e$ (theory of elements): arbitrary
- Axioms for read, write

\[
\begin{align*}
\text{read}(\text{write}(a, i, e), i) & \approx e \\
j \not\approx i \lor \text{read}(\text{write}(a, i, e), j) & = \text{read}(a, j).
\end{align*}
\]
Theories of arrays

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

**Theory of arrays** $T_{\text{arrays}}$:

- $T_i$ (theory of indices): Presburger arithmetic
- $T_e$ (theory of elements): arbitrary
- Axioms for read, write

\[
\forall a, i, e \quad \text{read}(\text{write}(a, i, e), i) \approx e
\]
\[
\forall a, i, j, e \quad j \not\approx i \lor \text{read}(\text{write}(a, i, e), j) \approx \text{read}(a, j).
\]

**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: $(x \leq 3 \lor x \approx y) \land y \leq z$ is an index guard

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

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

  $(\forall i)(\phi_I(i) \rightarrow \phi_V(i))$, where:

  - $\phi_I$: index guard
  - $\phi_V$: formula in which any universally quantified $i$ occurs in a direct array read; no nestings

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

  Example: $\forall x, y(x < y \rightarrow a[x] < a[y])$ is not an array property formula
Array Property Fragment

Definition (The Array Property Fragment) [Bradley,Manna,Sipma’06] The array property fragment consists of all existentially-closed Boolean combinations of array property formulae and quantifier-free $T_{\text{arrays}}$-formulae. The height of a formula in the fragment is the maximum height of an array property subformula.

Notation: $a[i] := \text{read}(a, i)$  $a\{k \leftarrow v\} := \text{write}(a, k, v)$

Example (Array Property Formula)
The following formula is in the array property fragment of $T_{\text{arrays}}$:

$(\exists a : \text{array})(\exists w, x, y, z, k, l, n : \text{index})w < x < y < z \land 0 < k < l < n \land l - k > 1$

$\land \text{sorted}(0, n - 1, a\{k \leftarrow w\}\{l \leftarrow x\}) \land \text{sorted}(0, n - 1, a\{k \leftarrow y\}\{l \leftarrow z\})$

where: $\text{sorted}(l, u, a)$ is the condition that the array $a$ is sorted (nondecreasing) between elements $l$ and $u$ and can be described by the formula:

$\forall i, j \ (l \leq i \leq j \leq u \rightarrow a[i] \leq a[j])$
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:

\[
\frac{F[\text{write}(a, i, v)]}{F[a'] \land a'[i] = v \land (\forall j. j \neq i \rightarrow a[j] = a'[j])}
\]

for fresh \(a'\) (write)

Given a formula \(F\) containing an occurrence of a write term \(\text{write}(a, i, v)\), we can substitute every occurrence of \(\text{write}(a, i, v)\) with a fresh variable \(a'\) and explain the relationship between \(a'\) and \(a\).
Step 3 Apply the following rule exhaustively to remove existential quantification:

\[
\frac{F[\exists i. G[i]]}{F[G[j]]}
\]

for fresh \( j \) (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.
Step 4 From the output F3 of Step 3, construct the index set $\mathcal{I}$:

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

(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 $\text{read}(a, t)$ 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.

$\lambda$ 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:

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

where \( n \) is the size of the list of quantified variables \( \vec{i} \).

This is the key step.

It replaces universal quantification with finite conjunction over the index set. The notation \( \vec{i} \in I^n \) means that the variables \( \vec{i} \) range over all \( n \)-tuples of terms in \( I \).
Theories of arrays

**Step 6:** From the output $F_5$ of **Step 5**, construct

\[ F_6 : \quad F_5 \land \bigwedge_{i \in I \setminus \{\lambda\}} \lambda \neq i \]

The new conjuncts assert that the variable $\lambda$ introduced in **Step 4** is unique: it does not equal any other index mentioned in $F_5$.

**Step 7:** Decide the TA-satisfiability of $F_6$ using the decision procedure for the quantifier free fragment.
Consider the array property formula

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

It contains one array property,

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

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

**Step 1:** The formula is already in NNF.

**Step 2:** We rewrite \( F \) as:

\[ F2 : a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\forall i. i \neq l \rightarrow a[i] = b[i]) \]
\[ \land a'[l] = v \land (\forall j. j \neq l \rightarrow a[j] = a'[j]). \]
Example

Consider the array property formula

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

Step 2: We rewrite \( F \) as:

\[ F_2 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\forall i. i \neq l \rightarrow a[i] = b[i]) \]
\[ \land a'[l] = v \land (\forall j. j \neq l \rightarrow a[j] = a'[j]). \]

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

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

Step 3: \( F_2 \) does not contain any existential quantifiers \( \iff F_3 = F_2 \).

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

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

Step 3:

\[ F_3 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\forall i. i \neq l \rightarrow a[i] = b[i]) \]
\[ \land a'[l] = v \land (\forall j. j \neq l \rightarrow a[j] = a'[j]). \]

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

Step 5: we replace universal quantification as follows:

\[ F_5 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land \bigwedge_{i \in \mathcal{I}} (i \neq l \rightarrow a[i] = b[i]) \]
\[ \land a'[l] = v \land \bigwedge_{i \in \mathcal{I}} (j \neq l \rightarrow a[j] = a'[j]). \]
Example

Consider the array property formula

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

\[ I = \{\lambda, k, l, l - 1, l + 1\} \]

Step 5 (continued) Expanding produces:

\[ F5' : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \land (k \neq l \rightarrow a[k] = b[k]) \land (l \neq l \rightarrow a[l] = b[l]) \land \]

\[ (l - 1 \neq l \rightarrow a[l - 1] = b[l - 1]) \land (l + 1 \neq l \rightarrow a[l + 1] = b[l + 1]) \land \]

\[ a'[l] = v \land (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \land (k \neq l \rightarrow a[k] = a'[k]) \land \]

\[ (l \neq l \rightarrow a[l] = a'[l]) \land (l - 1 \neq l \rightarrow a[l - 1] = a'[l - 1]) \land \]

\[ (l + 1 \neq l \rightarrow a[l + 1] = a'[l + 1]). \]
Example

Consider the array property formula

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

\[ I = \{\lambda\} \cup \{k\} \cup \{l, l-1, l+1\} = \{\lambda, k, l, l-1, l+1\} \]

Step 5 (continued): Simplifying produces

\[ F''_5 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \]

\[ \land (k \neq l \rightarrow a[k] = b[k]) \land a[l - 1] = b[l - 1] \land a[l + 1] = b[l + 1] \]

\[ \land a'[l] = v \land (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \]

\[ \land (k \neq l \rightarrow a[k] = a'[k]) \land a[l - 1] = a'[l - 1] \land a[l + 1] = a'[l + 1]. \]
Example

Consider the array property formula

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

Step 6 distinguishes \( \lambda \) from other members of \( I \):

\[ F6 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \]
\[ \land (k \neq l \rightarrow a[k] = b[k]) \land a[l - 1] = b[l - 1] \land a[l + 1] = b[l + 1] \]
\[ \land a'[l] = v \land (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \]
\[ \land (k \neq l \rightarrow a[k] = a'[k]) \land a[l - 1] = a'[l - 1] \land a[l + 1] = a'[l + 1] \]
\[ \land \lambda \neq k \land \lambda \neq l \land \lambda \neq l - 1 \land \lambda \neq l + 1. \]
Example

Consider the array property formula

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

Step 6 Simplifying, we have

\[ F'6 : \quad a'[k] = b[k] \land b[k] \neq v \land a[k] = v \land a[\lambda] = b[\lambda] \]
\[ \land a[k] = b[k] \land a[l - 1] = b[l - 1] \land a[l + 1] = b[l + 1] \]
\[ \land a'[l] = v \land a[\lambda] = a'[\lambda] \]
\[ \land (k \neq l \rightarrow a[k] = a'[k]) \land a[l - 1] = a'[l - 1] \land a[l + 1] = a'[l + 1] \]
\[ \land \lambda \neq k \land \lambda \neq l \land \lambda \neq l - 1 \land \lambda \neq l + 1. \]

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 $F_6$ of Step 6 is $T_{arrays}$-equisatisfiable to $F$.

**Proof**

*(Soundness)* Step 1-6 preserve satisfiability

($F_i$ 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 $F_6$ of Step 6 is $T_{\text{arrays}}$-equisatisfiable to $F$.

**Proof** (Completeness)

**Step 6:** From the output $F_5$ of Step 5, construct

$$F_6 : F_5 \land \bigwedge_{i \in \mathcal{I} \setminus \{\lambda\}} \lambda \neq i$$

Assume that $F_6$ is satisfiable. Clearly $F_5$ 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:

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

Assume that $F_5$ is satisfiable. Let $A = (\mathbb{Z}, \text{Elem}, \{a_A\}_{a \in \text{Arrays}}, \ldots)$ be a model for $F_5$. Construct a model $B$ for $F_4$ as follows.

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

$$a_B(x) = \begin{cases} 
a_A(l(x)) & \text{if } x - l(x) \leq u(x) - x \text{ or } u(x) = \infty \\
a_A(u(x)) & \text{if } x - l(x) > u(x) - x \text{ or } l(x) = -\infty
\end{cases}$$
Soundness and Completeness

**Theorem** (Soundness and Completeness)

Consider a formula $F$ from the array property fragment. The output $F_6$ 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 $F_3$ has model then $F_2$ has model
Soundness and Completeness

**Theorem** (Soundness and Completeness)

Consider a formula $F$ from the array property fragment. The output $F_6$ of Step 6 is $T_{\text{arrays}}$-equisatisfiable to $F$.

**Proof** (*Completeness*)

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

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

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

If $F_2$ has a model then $F_1$ has a model.

**Step 1:** Put $F$ in NNF: NNF $F_1$ is equivalent to $F$. 
Theories of arrays

**Theorem** (Complexity) Suppose \((T_{\text{index}} \cup T_{\text{elem}})\)-satisfiability is in NP. For sub-fragments of the array property fragment in which formulae have bounded-size blocks of quantifiers, \(T_{\text{arrays}}\)-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 \(n\).
Example: Does \texttt{BubbleSort} return a sorted array?

```c
int \[\] \texttt{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

Example: Does BubbleSort return a sorted array?

```c
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):` \( \forall i, j (l_1 \leq i \leq u_1 \leq l_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j]) \)
Example: Does BubbleSort return a sorted array?

```c
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, l1, u1, l2, u2): \( \forall i, j (l1 \leq i \leq u1 \leq l2 \leq j \leq u2 \rightarrow a[i] \leq a[j]) \)

To prove: \( C_2(a) \land \text{Update}(a, a') \rightarrow C_2(a') \)
Another Situation

Insertion of an element \( c \) in a sorted array \( a \) of length \( n \)

\[
\text{for } (i := 1; i \leq n; i := i + 1) \{ \\
\quad \text{if } a[i] \geq c \{ n := n + 1 \\
\quad \quad \text{for } (j := n; j > i; j := j - 1) \{ a[i] := a[i - 1] \} \\
\quad a[i] := c; \text{ return } a \\
\} \} a[n + 1] := c; \text{ return } a
\]

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

\[\text{Sorted}(a, n) \land \text{Update}(a, n, a', n') \land \neg\text{Sorted}(a', n') \models \top \]
Another Situation

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

\[ \text{Sorted}(a, n) \land \text{Update}(a, n, a', n') \land \neg \text{Sorted}(a', n') \models \top \perp? \]

\[
\begin{align*}
\text{Sorted}(a, n) & \quad \forall i, j(1 \leq i \leq j \leq n \rightarrow a[i] \leq a[j]) \\
\text{Update}(a, n, a', n') & \quad \forall i((1 \leq i \leq n \land a[i] < c) \rightarrow a'[i] = a[i]) \\
& \quad \forall i((c \leq a(1) \rightarrow a'[1] := c)) \\
& \quad \forall i((a[n] < c \rightarrow a'[n + 1] := c)) \\
& \quad \forall i((1 \leq i - 1 \leq i \leq n \land a[i - 1] < c \land a[i] \geq c) \rightarrow (a'[i] = c)) \\
& \quad \forall i((1 \leq i - 1 \leq i \leq n \land a[i - 1] \geq c \land a[i] \geq c \rightarrow a'[i] := a[i - 1])) \\
n' & \quad := n + 1 \\
\neg \text{Sorted}(a', n') & \quad \exists k, l(1 \leq k \leq l \leq n' \land a'k > a'[l])
\end{align*}
\]
Beyond the array property fragment

**Extension:** New arrays defined by case distinction – Def\((f')\)

\[
\forall \bar{x} (\phi_i(\bar{x}) \rightarrow f'(\bar{x}) = s_i(\bar{x})) \quad i \in I, \text{ where } \phi_i(\bar{x}) \land \phi_j(\bar{x}) \models T_0 \perp \text{ for } i \neq j (1)
\]

\[
\forall \bar{x} (\phi_i(\bar{x}) \rightarrow t_i(\bar{x}) \leq f'(\bar{x}) \leq s_i(\bar{x})) \quad i \in I, \text{ where } \phi_i(\bar{x}) \land \phi_j(\bar{x}) \models T_0 \perp \text{ for } i \neq j (2)
\]

where \(s_i, t_i\) are terms over the signature \(\Sigma\) such that \(T_0 \models \forall \bar{x} (\phi_i(\bar{x}) \rightarrow t_i(\bar{x}) \leq s_i(\bar{x}))\) for all \(i \in I\).

\(T_0 \subseteq T_0 \land \text{Def}(f')\) has the property that for every set \(G\) of ground clauses in which there are no nested applications of \(f'\):

\[
T_0 \land \text{Def}(f') \land G \models \perp \iff T_0 \land \text{Def}(f') [G] \land G
\]

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

- Some of the syntactic restrictions of the array property fragment can be lifted
Pointer Structures

[McPeak, Necula 2005]

• pointer sort $p$, scalar sort $s$; pointer fields ($p \rightarrow p$); scalar fields ($p \rightarrow s$);
• axioms: $\forall p \ E \lor C$; $E$ contains **disjunctions of pointer equalities**
  $C$ contains **scalar constraints**

**Assumption:** If $f_1(f_2(\ldots f_n(p)))$ occurs in axiom, the axiom also contains:

$p=\text{null} \lor f_n(p)=\text{null} \lor \cdots \lor f_2(\ldots f_n(p)))=\text{null}$

**Example:** doubly-linked lists; ordered elements

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

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

$\forall p \ (p \neq \text{null} \land p.\text{next} \neq \text{null} \rightarrow p.\text{info} \leq p.\text{next}.\text{info})$
Pointer Structures

[McPeak, Necula 2005]

- pointer sort \( p \), scalar sort \( s \); pointer fields \( (p \rightarrow p) \); scalar fields \( (p \rightarrow s) \);
- axioms: \( \forall p \ E \lor C \); \( E \) contains disjunctions of pointer equalities
  \( C \) contains scalar constraints

Assumption: If \( f_1(f_2(\ldots f_n(p)))) \) occurs in axiom, the axiom also contains:
\( p=null \lor f_n(p)=null \lor \cdots \lor f_2(\ldots f_n(p)))=null \)

Theorem. \( K \) set of clauses in the fragment above. Then for every set \( G \) of
ground clauses, \((K \cup G) \cup T_s \models \bot\) iff \( K^{[G]} \cup T_s \models \bot\)
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

∀p (p ≠ null ∧ p.next ≠ null → p.next.prev = p)
∀p (p ≠ null ∧ p.prev ≠ null → p.prev.next = p)
∧ c≠null ∧ c.next≠null ∧ d≠null ∧ d.next≠null ∧ c.next=d.next ∧ c ≠ d  ⊨ ⊥
Example: A theory of doubly-linked lists

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

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

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

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

$(c.\text{next} = d.\text{next} \land c \neq d) \models \bot$
Example: List insertion

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

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

for all $p \neq c$ do

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

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

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

$p.next \neq \text{null} \land p.next.prio \leq x$ then $p.next' = c$, $c.next' = p.next$

Verification task: After insertion list remains sorted
Example: List insertion

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

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

 for all $p \neq c$ do

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

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

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

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

 Verification task: After insertion list remains sorted
Example: List insertion

Initially list is sorted: \( p\).next \(\neq\) null \(\rightarrow\) \( p\).prio \(\geq\) \( p\).next.prio

\( c\).prio = \( x\), \( c\).next = null

for all \( p \neq c \) do

if \( p\).prio \(\leq\) \( x\) then if First\((p)\) then \( c\).next' = \( p\), First'\((c)\), ¬First'\((p)\) endif; \( p\).next' = \( p\).next

\( p\).prio > \( x\) then case \( p\).next = null then \( p\).next' := \( c\), \( c\).next' = null

\( p\).next \(\neq\) null \(\land\) \( p\).next.prio > \( x\) then \( p\).next' = \( p\).next

\( p\).next \(\neq\) null \(\land\) \( p\).next.prio \(\leq\) \( x\) then \( p\).next' = \( c\), \( c\).next' = \( p\).next

Verification task: After insertion list remains sorted
Example: List insertion

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

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

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

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

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

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

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

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

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

To check: $\text{Sorted}(\text{next}, \text{prio}) \land \text{Update}(\text{next}, \text{next}') \land p_0.\text{next}' \neq \text{null} \land p_0.\text{prio} \not\geq p_0.\text{next}'.\text{prio} \models \bot$
To show:
\[ \mathcal{T}_2 \cup \neg \text{Sorted}(\text{next}') \models \bot \]

\[ \mathcal{T}_2 = \mathcal{T}_1 \cup \text{Update}(\text{next, next}') \]

\[ \mathcal{T}_1 = \mathcal{T}_0 \cup \text{Sorted}(\text{next}) \]

\[ \mathcal{T}_0 = (\text{Lists, next}) \]
Example: List insertion

To show:

\[ T_2 = T_1 \cup \text{Update}(\text{next}, \text{next}') \]

\[ T_2 \cup \neg \text{Sorted}(\text{next}') \models \bot \]

Instantiate:

\[ T_1 \cup \text{Update}(\text{next}, \text{next}')[G] \cup G \models \bot \]

Hierarchical reasoning:

\[ T_1 = T_0 \cup \text{Sorted}(\text{next}) \]

\[ T_0 = (\text{Lists}, \text{next}) \]
Example: List insertion

\[ \mathcal{T}_2 = \mathcal{T}_1 \cup \text{Update}(\text{next}, \text{next}') \]

\[ \mathcal{T}_1 = \mathcal{T}_0 \cup \text{Sorted}(\text{next}) \]

\[ \mathcal{T}_0 = (\text{Lists, next}) \]

To show:

\[ \mathcal{T}_2 \cup \neg \text{Sorted}(\text{next}') \models \perp \]

\[ \mathcal{T}_1 \cup G'(\text{next}) \models \perp \]

\[ \mathcal{T}_0 \cup G'' \models \perp \]
More general concept

Local Theory Extensions
Satisfiability of formulae with quantifiers

**Goal:** generalize the ideas for extensions of theories
Example: Strict monotonicity

\[ \mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot \]

\[ \text{Mon}(f) \quad \forall i, j (i < j \rightarrow f(i) < f(j)) \]

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 (\( \text{Mon}, \mathbb{Z}, \mathbb{R} \): non-disjoint signatures)
- SMT provers may have problems with the universal quantifiers

Our goal: reduce search: consider certain instances \( \text{Mon}(f)[G] \)
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 $G$,
$\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$ 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

\[
\mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot
\]

<table>
<thead>
<tr>
<th>Base theory ((\mathbb{R} \cup \mathbb{Z}))</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a &lt; b)</td>
<td>(f(a) = f(b) + 1)</td>
</tr>
<tr>
<td></td>
<td>(\forall i, j (i &lt; j \rightarrow f(i) &lt; f(j)))</td>
</tr>
</tbody>
</table>
Example: Strict monotonicity

\[
\mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot
\]

Extension is local \(\mapsto\) replace axiom with ground instances

<table>
<thead>
<tr>
<th>Base theory ((\mathbb{R} \cup \mathbb{Z}))</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a &lt; b)</td>
<td>(f(a) = f(b) + 1)</td>
</tr>
<tr>
<td></td>
<td>(a &lt; b \rightarrow f(a) &lt; f(b))</td>
</tr>
<tr>
<td></td>
<td>(b &lt; a \rightarrow f(b) &lt; f(a))</td>
</tr>
</tbody>
</table>

Solution 1: \(SMT(\mathbb{R} \cup \mathbb{Z} \cup \text{UIF})\)
Example: Strict monotonicity

\[ \mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot \]

Extension is local \iff replace axiom with ground instances

Add congruence axioms. Replace pos-terms with new constants

Solution 2: Hierarchical reasoning

<table>
<thead>
<tr>
<th>Base theory ((\mathbb{R} \cup \mathbb{Z}))</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a &lt; b)</td>
<td>(f(a) = f(b) + 1)</td>
</tr>
<tr>
<td></td>
<td>(a &lt; b \rightarrow f(a) &lt; f(b))</td>
</tr>
<tr>
<td></td>
<td>(b &lt; a \rightarrow f(b) &lt; f(a))</td>
</tr>
<tr>
<td></td>
<td>(a = b \rightarrow f(a) = f(b))</td>
</tr>
</tbody>
</table>

\text{Solution 2: Hierarchical reasoning}
### Example: Strict monotonicity

\[ \mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot \]

Extension is local \(\mapsto\) replace axiom with ground instances

Replace \(f\)-terms with new constants

Add definitions for the new constants

<table>
<thead>
<tr>
<th>Base theory ((\mathbb{R} \cup \mathbb{Z}))</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a &lt; b)</td>
<td>(a_1 = b_1 + 1)</td>
</tr>
<tr>
<td></td>
<td>(a &lt; b \rightarrow a_1 &lt; b_1)</td>
</tr>
<tr>
<td></td>
<td>(b &lt; a \rightarrow b_1 &lt; a_1)</td>
</tr>
<tr>
<td></td>
<td>(a = b \rightarrow a_1 = b_1)</td>
</tr>
</tbody>
</table>
Example: Strict monotonicity

\[ \mathbb{R} \cup \mathbb{Z} \cup \text{Mon}(f) \cup (a < b \land f(a) = f(b) + 1) \models \bot \]

Extension is local ⇔ replace axiom with ground instances

Replace \( f \)-terms with new constants

Add definitions for the new constants

<table>
<thead>
<tr>
<th>Base theory ( (\mathbb{R} \cup \mathbb{Z}) )</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a &lt; b )</td>
<td>( a_1 = f(a) )</td>
</tr>
<tr>
<td>( a_1 = b_1 + 1 )</td>
<td>( b_1 = f(b) )</td>
</tr>
<tr>
<td>( a &lt; b \rightarrow a_1 &lt; b_1 )</td>
<td></td>
</tr>
<tr>
<td>( b &lt; a \rightarrow b_1 &lt; a_1 )</td>
<td></td>
</tr>
<tr>
<td>( a = b \rightarrow a_1 = b_1 )</td>
<td></td>
</tr>
</tbody>
</table>
Reasoning in local theory extensions

**Locality:** \[ T_0 \cup K \cup G \models \bot \quad \text{iff} \quad T_0 \cup K[G] \cup G \models \bot \]

**Problem:** Decide whether \( T_0 \cup K[G] \cup G \models \bot \)

**Solution 1:** Use \( SMT(T_0 + UIF) \): possible only if \( K[G] \) ground

**Solution 2:** Hierarchic reasoning [VS’05]

reduce to satisfiability in \( T_0 \): applicable in general

\[ \rightarrow \text{parameterized complexity} \]
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: \( 0 \leq \text{min} < \text{max} \) \( \mathbb{R} \)

Minimum secure distance: \( l_{\text{alarm}} > 0 \) \( \mathbb{R} \)

Time between updates: \( \Delta t > 0 \) \( \mathbb{R} \)

Train positions before and after update: \( pos(i), \text{pos}'(i) : \mathbb{Z} \rightarrow \mathbb{R} \)
Example

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

Update \((pos, pos')\):

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

...
Example

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

Inductive invariant:  
\[ \text{Safe}(\text{pos}) \land \text{Update}(\text{pos}, \text{pos}') \land \neg \text{Safe}(\text{pos}') \models \mathcal{T}_S \bot \]

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

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

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

\[ \mathcal{T}_0 = \mathbb{R} \cup \mathbb{Z} \]

To show:

\[ \mathcal{T}_2 \cup \neg \text{Safe}(\text{pos}') \models \bot \ (G) \]
Example

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

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

\[ \mathcal{T}_0 = \mathbb{R} \cup \mathbb{Z} \]

\[ \Phi(c, c_{\text{pos}'}, d_{\text{pos}}, n, l_{\text{alarm}}, \text{min}, \text{max}, \Delta t) \models \perp \]

To show:

\[ \mathcal{T}_2 \cup \neg \text{Safe}(\text{pos}') \models \perp \]

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

\[ \mathcal{T}_0 \cup G'' \models \perp \]

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 \rightarrow Transition constr. \rightarrow Decision procedures (H-PILoT)
Example 2: Parametric topology

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

**Assumptions:**
- No cycles
- in-degree (out-degree) of associated graph at most 2.
**Parametricity and modularity**

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

Assumptions:
- No cycles
- In-degree (out-degree) of associated graph at most 2.

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

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

Assumptions:
- No cycles
- In-degree (out-degree) of associated graph at most 2.

Data structures:
- \( p_1 \): trains
- \( p_2 \): segments
- 2-sorted pointers
- 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

\begin{align*}
\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 \land \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 \land \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t) & \rightarrow \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0)
\end{align*}
Incoming and outgoing trains
Incoming and outgoing trains

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

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

$t \neq t_1$, ids(segm($t$)) $<$ ids($s_1$), next$_t(t)$ $=$ null$_t$, alloc($s_1$) $=$ tid($t_1$) $\rightarrow$ next’($t$) $=$ $t_1$ $\land$ next’($t_1$) $=$ null$_t$

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

...
Incoming and outgoing trains
Safety property

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

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

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant (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,

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

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
Further extensions (Systems of LHA)

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

Sensors + Communication Channels

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

Collision free: $\forall i, j (\text{lane}(i) = \text{lane}(j) \wedge \text{pos}(i) \geq \text{pos}(j) \wedge i \neq j \rightarrow \text{pos}(i) - \text{pos}(j) > d)$
Model families \( \{ S(i) \mid i \in I \} \) consisting of an unbounded number of similar interacting systems.

- Model the interaction
- Model the systems \( S(i) \)
- Model the topology updates
Model: Families of similar interacting systems

Model families \( \{ S(i) \mid i \in I \} \) consisting of an unbounded number of similar interacting systems.

- **Model the interaction**

\[ P = P_S \cup P_N \]

\[ \text{The functions in } P \text{ model the way the systems perceive their neighbors} \]

\( P_S \text{ sensors:} \)

- \( \text{sideback}(7) = 3 \)
- \( \text{back}(7) = 3 \)
- \( \text{front}(7) = \text{nil} \)
- \( \text{sidefront}(7) = 10 \)

\( P_N: \text{neighborhood links} \)

\[ f_1, f_2, f_3, f_4 \]
Model: Families of similar interacting systems

Model families \(\{S(i) \mid i \in I\}\) consisting of an unbounded number of similar interacting systems.

- Model the interaction \(\mapsto\) structures \((I, \{p : I \to I\}_{p \in P})\)
- Model the systems \(S(i)\) \(\mapsto\) hybrid automata
Model: Spatial families of LHA

Model families \( \{S(i) \mid i \in I\} \) consisting of an unbounded number of similar interacting systems.

- Model the interaction \( \mapsto \) structures (\( I, \{ p : I \to I \}_{p \in P} \))
- Model the systems \( S(i) \) \( \mapsto \) hybrid automata
- Model the topology updates \( \mapsto \) Topology automaton

Example: Update(front, front')

\[ \forall i (i \neq \text{nil} \land \text{Prop}(i) \land \neg \exists j (\text{ASL}(j, i)) \rightarrow \text{front}'(i) = \text{nil}) \]

\[ \forall i (i \neq \text{nil} \land \text{Prop}(i) \land \exists j (\text{ASL}(j, i)) \rightarrow \text{Closest}_f (\text{front}'(i), i)) \]

\[ \forall i (i \neq \text{nil} \land \neg \text{Prop}(i) \rightarrow \text{front}'(i) = \text{front}(i)) \]

\( \text{ASL}(j, i): \quad j \neq \text{nil} \land \text{lane}(j) = \text{lane}(i) \land \text{pos}(j) > \text{pos}(i) \quad j \text{ is ahead of } i \text{ on the same lane} \)

\( \text{Closest}_f (j, i): \quad \text{ASL}(j, i) \land \forall k (\text{ASL}(k, i) \rightarrow \text{pos}(k) \geq \text{pos}(j)) \quad j \text{ is ahead of } i; \text{ no car between them.} \)
Verification

Is safety property an inductive invariant?
Verification

Is safety property an inductive invariant?

Local extensions: use H-PILoT

- Unsatisfiable $\iff$ Safety invariant
- Satisfiable $\iff$ Model
Verification

Is safety property an inductive invariant?

Local extensions: use \textbf{H-PILoT}

- Unsatisfiable $\mapsto$ Safety invariant
- Satisfiable $\mapsto$ Model $\mapsto$ Simulation [J. Wild, BSc Thesis 2018]
Other interesting topics

• Generate invariants

• Verification by abstraction/refinement
Abstraction-based Verification

\[ \phi(1) \land Tr(1, 2) \land \cdots \land Tr(n - 1, n) \land \neg \text{safe}(n) \]

- 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:
  Every summer (usually end of August):
    Summer school “Verification Technology, Systems & Applications”

BSc/MSc Theses in the area