Decision Procedures in Verification

Combinations of Decision Procedures (4)

30.1.2014

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now:

Decidable subclasses of FOL

Decision procedures for single theories

Uninterpreted function symbols

Decision procedures for numeric domains

Combinations of theories

The Nelson-Oppen combination procedure.

Beyond the conjunctive fragment $DPLL(\mathcal{T})$

Checking satisfiability of quantified formulae

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.

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

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

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

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

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

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

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:

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

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 $\mathcal{I}$:

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

(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:

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

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

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

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.

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

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

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:

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

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

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}
```

$-1 \le i < |a| \wedge$
partitioned$(a, 0, i, i+1, |a|-1) \wedge$
sorted$(a, i, |a|-1)$

$-1 \le i < |a| \wedge 0 \le j \le i \wedge$
partitioned$(a, 0, i, i+1, |a|-1) \wedge$
sorted$(a, i, |a|-1)$
partitioned$(a, 0, j-1, j, j)$ C_2

Generate verification conditions and prove that they are valid

Predicates:

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

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

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

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.

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

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

Beyond the array property fragment

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

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

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

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 $i \in I$.

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

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

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

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

Pointer Structures

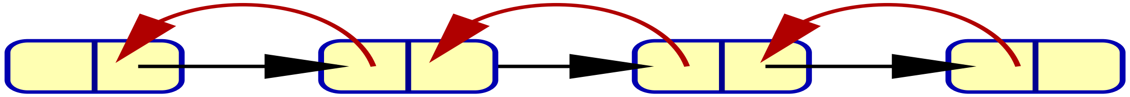
Pointer Structures

[McPeak, Necula 2005]

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

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

Example: doubly-linked lists; ordered elements



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

Pointer Structures

[McPeak, Necula 2005]

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

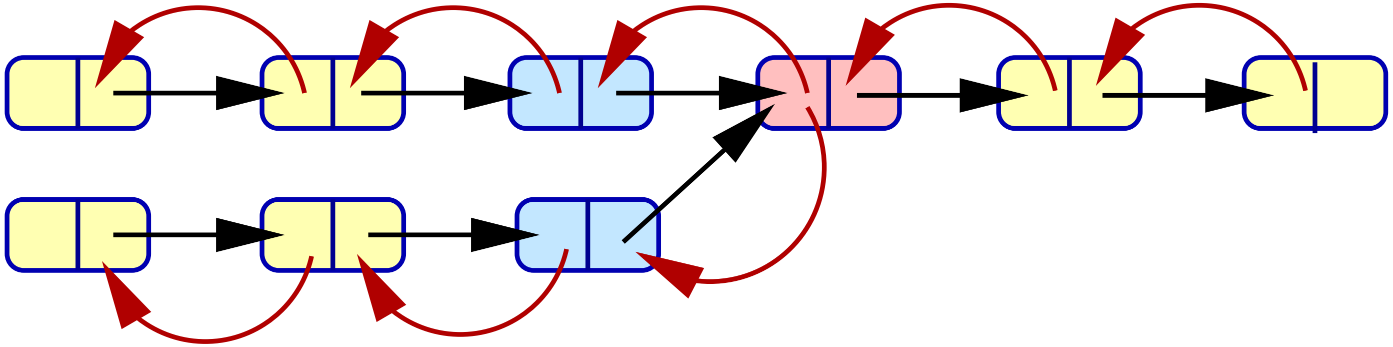
Assumption: If $f_1(f_2(\dots(f_n(p)))$ occurs in axiom, the axiom also contains:

$$p{=}\text{null} \vee f_n(p){=}\text{null} \vee \cdots \vee f_2(\dots f_n(p))){=}\text{null}$$

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

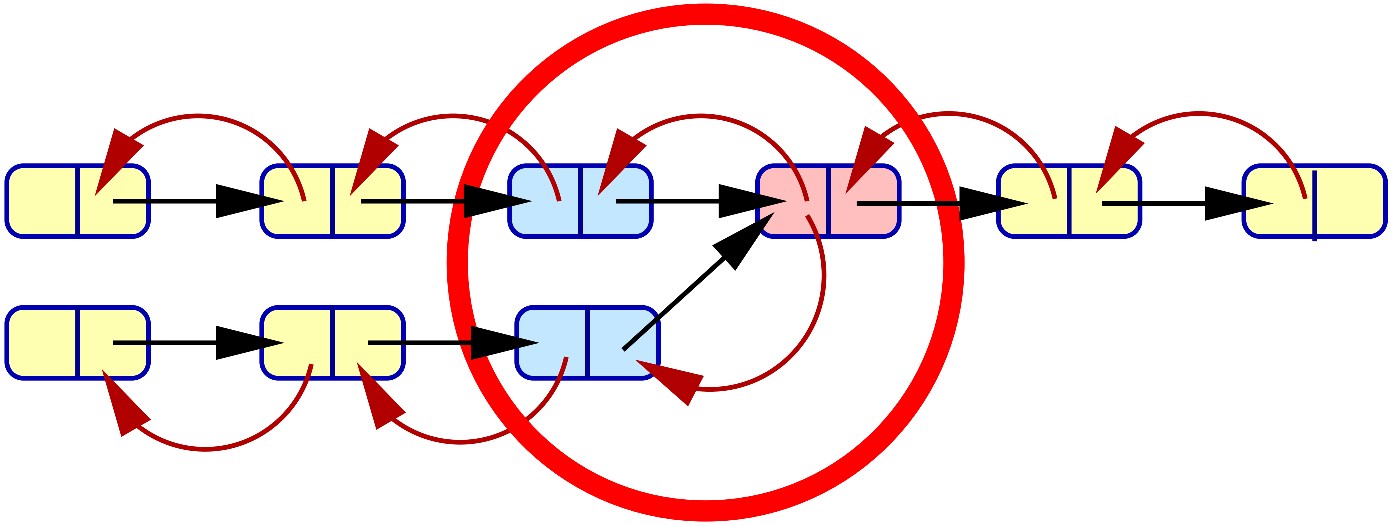


$$\forall p\ (p \neq \mathrm{null} \wedge p.\mathrm{next} \neq \mathrm{null}\ \ \rightarrow\ \ p.\mathrm{next}.\mathrm{prev} = p)$$

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

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

Example: A theory of doubly-linked lists

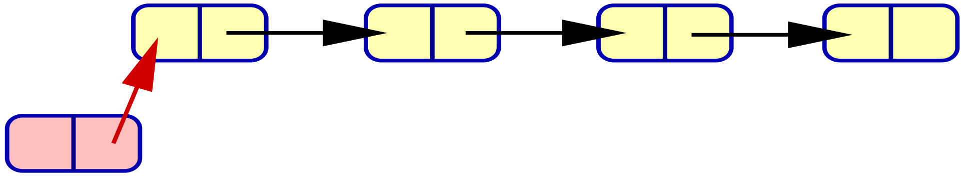


$(c \neq \text{null} \wedge c.\text{next} \neq \text{null} \rightarrow c.\text{next}.\text{prev} = c)$ $(c.\text{next} \neq \text{null} \wedge c.\text{next}.\text{next} \neq \text{null} \rightarrow c.\text{next}.\text{next}.\text{prev} = c.\text{next})$

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

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

Example: List insertion

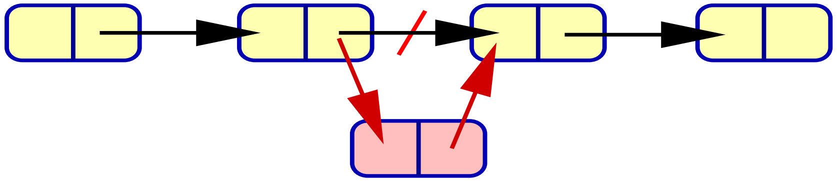


Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next}.\text{prio}$

```
$c.\text{prio} = x, c.\text{next} = \text{null}$
**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** $p.\text{next}' = c,\ c.\text{next}' = p.\text{next}$
```

Verification task: After insertion list remains sorted

Example: List insertion

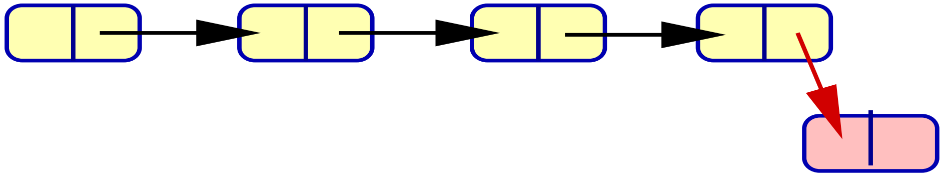


Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next}.\text{prio}$

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

Verification task: After insertion list remains sorted

Example: List insertion

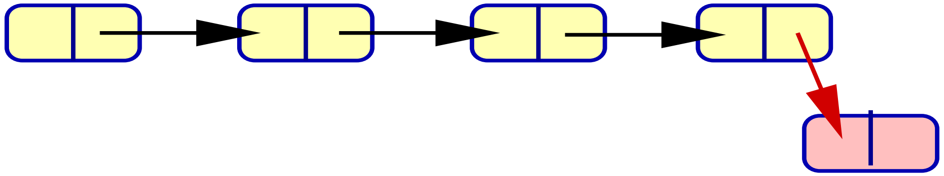


Initially list is sorted: $p.\text{next} \neq \text{null} \rightarrow p.\text{prio} \geq p.\text{next.prio}$

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

Verification task: After insertion list remains sorted

Example: List insertion



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

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

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

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

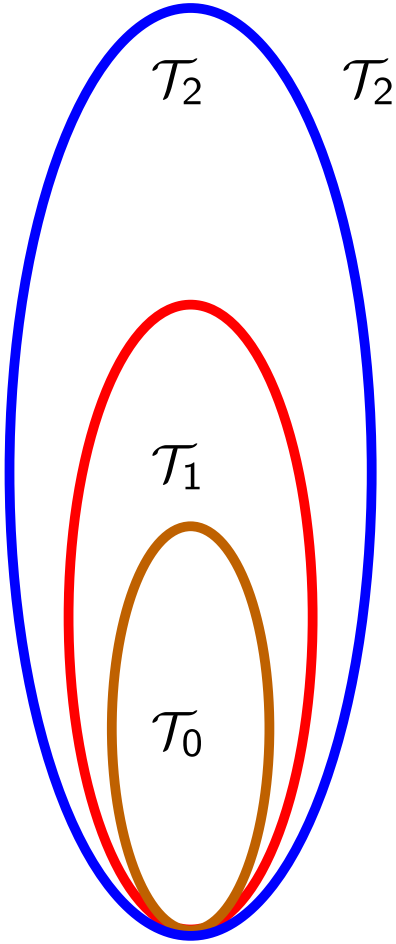
$\forall p(p{\neq}\mathrm{null} \wedge p{\neq}c \wedge \mathrm{prio}(p){>}x \wedge \mathrm{next}(p){\neq}\mathrm{null} \wedge \mathrm{prio}(\mathrm{next}(p)){>}x \rightarrow \mathrm{next}'(p){=}\mathrm{next}(p))$

$\forall p(p{\neq}\mathrm{null} \wedge p$ … $(p){=}c$
$\forall p(p{\neq}\mathrm{null} \wedge p$ … $(c){=}\mathrm{next}(p))$

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

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

Example: List insertion



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

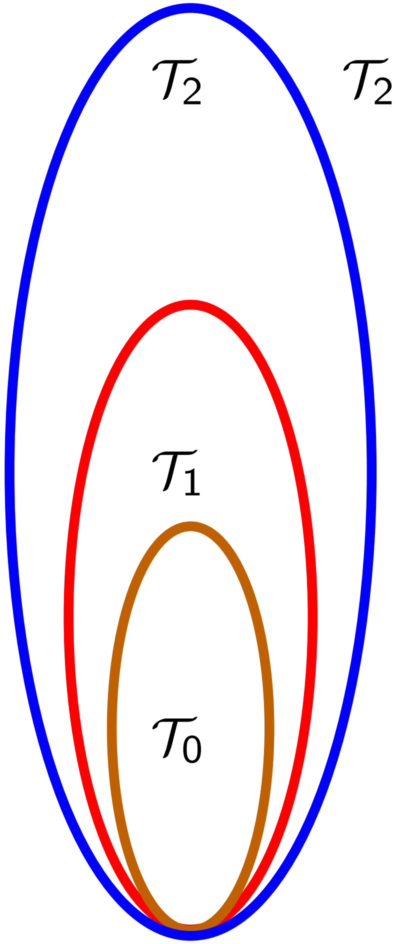
$\mathcal{T}_1 = \mathcal{T}_0 \cup \mathsf{Sorted}(\mathsf{next})$

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

To show:

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

Example: List insertion



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

To show:

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

Instantiate:

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

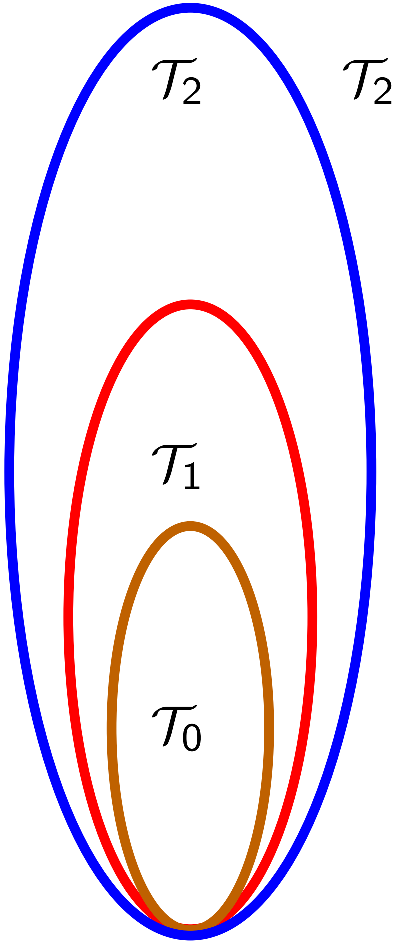
Hierarchical reasoning:

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

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

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

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

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

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

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 $\mathsf{Mon}(f)[G]$
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

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

Extension is local ↦ 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

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

Extension is local ↦ 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 ↦ 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

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

Extension is local ↦ 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

Hierarchical reasoning

Theorem: Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local. The following are equivalent:

(1) $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable

(2) $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ has a (partial) model in which all terms in G are defined

(3) $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0 \cup \mathrm{Con}[G]_0$ has a (total) model, where $\mathrm{Con}[G]_0$ is the set of instances of the congruence axioms corresponding to D:

$$\mathrm{Con}[G]_0 = \{ \bigwedge_{i=1}^{n} c_i = d_i \rightarrow c = d \mid f(c_1, \dots, c_n) = c, f(d_1, \dots, d_n) = d \in D \}$$

($\mathcal{K}_0 \cup G_0 \cup D$ be obtained from $\mathcal{K}[G] \cup G$ by purification)

Consequence: Hierarchical reduction to a satisfiability test in $\mathcal{T}_0$

Hierarchical reasoning

Theorem: Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local. The following are equivalent:

(1) $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable

(2) $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ has a (parti

(3) $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0 \cup \mathrm{Con}[G]_0$

of instances of the congr

$$\mathrm{Con}[G]_0 = \{ \bigwedge_{i=1}^{n} c_i = d_i \rightarrow$$

($\mathcal{K}_0 \cup G_0 \cup D$ be obtained from $\mathcal{K}[G] \cup G$ by purification)

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

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

Consequence: Hierarchical reduction to a satisfiability test in $\mathcal{T}_0$

Hierarchical reasoning

Theorem: Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local. The following are equivalent:

(1) $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable

(2) $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ has a (parti

(3) $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}[G]_0$
of instances of the congr

$$\mathsf{Con}[G]_0 = \{ \bigwedge_{i=1}^{n} c_i = d_i \rightarrow$$

($\mathcal{K}_0 \cup G_0 \cup D$ be obtaine

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

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

Consequence: Hierarchical reduction to a satisfiability test in $\mathcal{T}_0$

Hierarchical reasoning

Theorem: Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local. The following are equivalent:

(1) $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is satisfiable;

(2) $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ has a (parti

(3) $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0 \cup \text{Con}[G]_0$

of instances of the congr

$$\text{Con}[G]_0 = \{ \bigwedge_{i=1}^{n} c_i = d_i \rightarrow c$$

($\mathcal{K}_0 \cup G_0 \cup D$ be obtaine

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

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

Consequence: Hierarchical reduction to a satisfiability test in $\mathcal{T}_0$.

Recognizing local theory extensions

Problem: Determine whether a theory extension is local

Solutions:

1. **Semantic method:** Embeddability of partial models into total models

$$\mathcal{T}_1 \text{ local extension of } \mathcal{T}_0 \quad \rightleftarrows \quad \text{Emb}(\mathcal{T}_0, \mathcal{T}_1)$$

2. **Proof theoretical method:** Test saturation under ordered resolution
   [Basin,Ganzinger'96,'01] test locality; generate local presentation if poss.

Recognizing local theory extensions

Problem: Determine whether a theory extension is local

Our solutions:

1. **Semantic method:** Embeddability of partial models into total models

Results:

- Extensions with new functions +
  - definitions [VS'05,'06]
  - (piecewise) boundedness/monotonicity [VS, Ihlemann'07]
  - injectivity, strict monotonicity (add. asmpts.)[Jacobs,VS'07]
  - Lipschitz conds./continuity/derivability [VS'08]
- Theories of data structures [Ihlemann,Jacobs,VS'08]

Examples of local theory extensions

1. Monotonicity conditions

Theorem Any extension of the (i) theory of reals, rationals or integers or (ii) the theory of Posets, (semi)lattices, distributive lattices, Boolean algebras with functions satisfying $\mathsf{Mon}^\sigma(f)$ is local.

$$\mathsf{Mon}^\sigma(f) \quad \bigwedge_{i \in I} x_i {\leq_i}^{\sigma_i} y_i \wedge \bigwedge_{i \notin I} x_i{=}y_i \rightarrow f(x_1, .., x_n) \leq f(y_1, .., y_n)$$

Theorem. The extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathsf{SMon}(f)$ is local if $\mathcal{T}_0$ is the theory of reals (and $f : \mathbb{R} \rightarrow \mathbb{R}$) or the disjoint combination of the theories of reals and integers (and $f : \mathbb{Z} \rightarrow \mathbb{R}$).

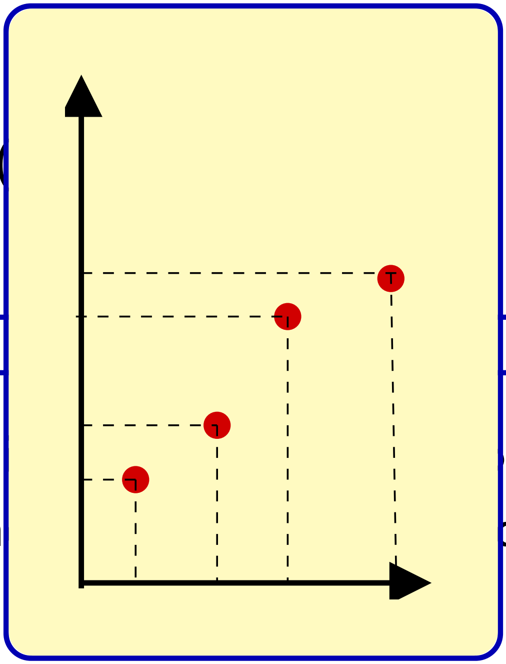
$$\mathsf{SMon}(f) \quad \forall i, j (i{<}j \rightarrow f(i){<}f(j))$$

Examples of local theory extensions

1. Monotonicity conditions

Theorem Any extension of the (i) theory of reals, rationals or integers or (ii) the theory of Posets, (semi)lattices, distributive lattices, Boolean algebras with functions satisfying $\mathrm{Mon}^{\sigma}(f)$ is local.

$$\mathrm{Mon}^{\sigma}(f) \quad \bigwedge_{i\in I} x_i {\leq_i}^{\sigma_i} y_i \wedge \bigwedge_{i\notin I} x_i = y_i \rightarrow f(x_1, .., x_n) \leq f($$



Theorem. The extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathrm{SMon}(f)$ is local f reals (and $f : \mathbb{R} \rightarrow \mathbb{R}$) or the disjoint combination of th d integers (and $f : \mathbb{Z} \rightarrow \mathbb{R}$).

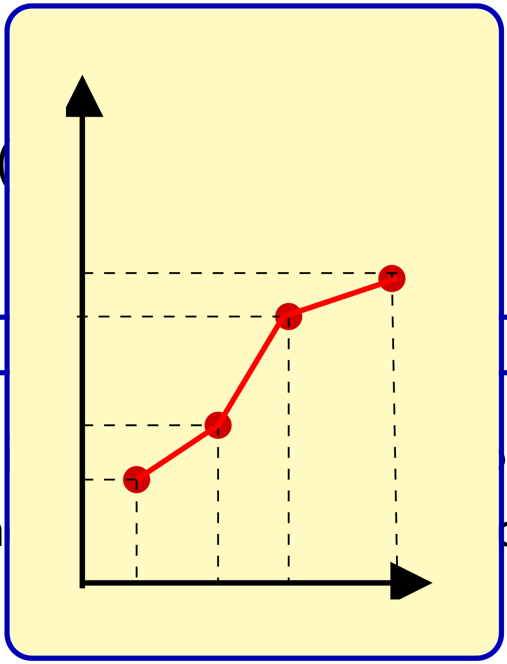
$$\mathrm{SMon}(f) \quad \forall i, j (i < j \rightarrow f(i) < f(j))$$

Examples of local theory extensions

1. Monotonicity conditions

Theorem Any extension of the (i) theory of reals, rationals or integers or (ii) the theory of Posets, (semi)lattices, distributive lattices, Boolean algebras with functions satisfying $\text{Mon}^{\sigma}(f)$ is local.

$$\text{Mon}^{\sigma}(f) \quad \bigwedge_{i \in I} x_i {\leq_i}^{\sigma_i} y_i \wedge \bigwedge_{i \notin I} x_i = y_i \rightarrow f(x_1, .., x_n) \leq f($$



Theorem. The extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \text{SMon}(f)$ is local f
reals (and $f : \mathbb{R} \rightarrow \mathbb{R}$) or the disjoint combination of th d
integers (and $f : \mathbb{Z} \rightarrow \mathbb{R}$).

$$\text{SMon}(f) \quad \forall i, j (i < j \rightarrow f(i) < f(j))$$

Examples of local theory extensions

2. Boundedness

Theorem. $\mathcal{T}_0$ contains reflexive binary predicate $\leq$, and $f \not\in \Sigma_0$.

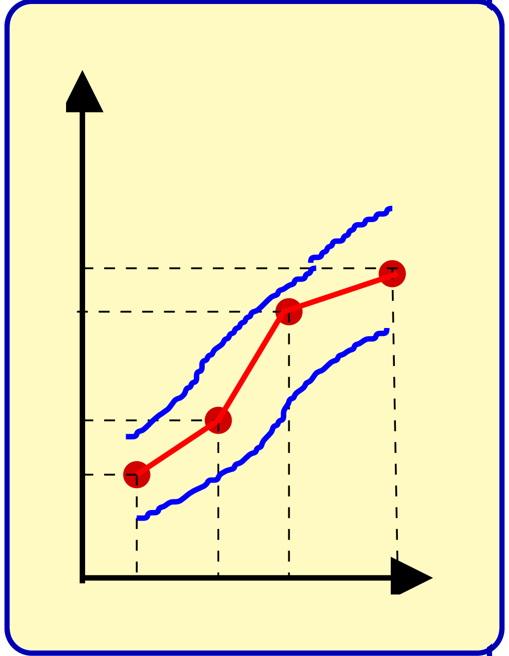
$t_1, \ldots t_m, s_1, \ldots s_m$: Σ_0-terms; $\phi_1, \ldots \phi_m$: Π_0-formulae s.t.

(i) $\mathcal{T}_0 \models \forall \overline{x}(\phi_i(\overline{x}) \rightarrow s_i(\overline{x}) \leq t_i(\overline{x}))$;

(ii) if $i \neq j$, $\phi_i \wedge \phi_j \models_{\mathcal{T}_0} \perp$.

$$\text{GB}(f) = \begin{cases} \forall \overline{x}(\phi_1(\overline{x}) \rightarrow s_1(\overline{x}) \leq f(\overline{x}) \leq t_1(\overline{x})) \\ \quad \ldots \\ \forall \overline{x}(\phi_m(\overline{x}) \rightarrow s_m(\overline{x}) \leq f(\overline{x}) \leq t_m(\overline{x})) \end{cases}$$

$$\text{Def}(f) = \begin{cases} \forall \overline{x}(\phi_1(\overline{x}) \rightarrow f(\overline{x}) = t_1(\overline{x})) \\ \quad \ldots \\ \forall \overline{x}(\phi_m(\overline{x}) \rightarrow f(\overline{x}) = t_m(\overline{x})) \end{cases}$$



The extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \text{GB}(f)$ and $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \text{Def}(f)$ are both local.

Examples of local theory extensions

2. Boundedness for monotone functions

Theorem. Any extension of a theory for which $\leq$ is a partial order (or at least reflexive) with functions satisfying $\mathsf{Mon}^\sigma(f)$ and $\mathsf{Bound}^t(f)$ is local.

$$\mathsf{Bound}^t(f) \qquad \forall x_1, \dots, x_n (f(x_1, \dots, x_n) \leq t(x_1, \dots, x_n))$$

where $t(x_1, \dots, x_n)$ is a Π_0-term whose associated function has the same monotonicity as f in any model.

Similar results hold for strictly monotone functions.

Applications

The notion of locality allows us to:

- uniformly explain existing results, e.g.
  - Local pointer structures [McPeak, Necula 2005]
  - Theory of arrays [Bradley,Manna,Sipma'06]
- generate / recognize in a systematic way a class of local theory extensions related to data structures, including proper extensions of the theories above.

e.g.:

- Updates of arrays, properties of arrays
- Insertion/Deletion in pointer structures