Decidable logics combining heap structures and data

P. Madhusudan   Gennaro Parlato   Xiaokang Qiu

Department of Computer Science
University of Illinois at Urbana-Champaign

MVD’10 - Midwest Verification Day
Outline

Overview

STRAND Logic

Deciding STRAND Fragments

Program Verification using STRAND
Logic and SMT Solvers

- Logic
  - a fundamental technique in program verification and analysis
  - many tools need some form of symbolic reasoning
  - high computational complexity
Logic and SMT Solvers

• Logic
  • a fundamental technique in program verification and analysis
  • many tools need some form of symbolic reasoning
  • high computational complexity

• SMT Solvers
  • check satisfiability in particular theories
  • engines of proof that serve many programming verification and analysis techniques
Applications

- Test case generation
- Verifying compilers
- Abstraction
- Invariant generation
- Type checker
- Model based testing
Applications@Microsoft of Z3
Theories Supported by SMT Solvers

- Equality over uninterpreted function and predicate symbols
- Real and integer arithmetic
- Bit-vectors
- Arrays
- Tuple/record/enumeration types and algebraic data-types
Theories Supported by SMT Solvers

- Equality over uninterpreted function and predicate symbols
- Real and integer arithmetic
- Bit-vectors
- Arrays
- Tuple/record/enumeration types and algebraic data-types

Heap Structure + Data?
(Example: binary search tree)
Related Work

HAVOC (*Lahiri and Qadeer: POPL’08*)
- reasoning with generic heaps combined with an arbitrary data-logic
- awkward syntax restrictions, to obtain decidability
- efficient, but not expressive, cannot even handle doubly-linked lists

CSL (*Bouajjani et al.: CONCUR’09*)
- similar sort-based syntax restrictions
- generalize to handle doubly-linked list
Related Work

HAVOC (*Lahiri and Qadeer: POPL’08*)
- reasoning with generic heaps combined with an arbitrary data-logic
- awkward syntax restrictions, to obtain decidability
- efficient, but not expressive, cannot even handle doubly-linked lists

CSL (*Bouajjani et al.: CONCUR’09*)
- similar sort-based syntax restrictions
- generalize to handle doubly-linked list

Both cannot even handle binary search trees!
Our Contribution

- A new logic, called STRAND
  - defined over a class of recursive data structure \( \mathcal{R} \)
  - \( \exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y}) \)
  - where \( \varphi \) is an Monadic Second Order (MSO) formula combines heap structures and data, but where the data-constraints are only allowed to refer to \( \vec{x} \) and \( \vec{y} \)
  - Example:
    \[
    \forall y_1 \forall y_2. (\exists z. (y_1 \rightarrow z \land z \rightarrow y_2) \Rightarrow (\text{data}(y_1) \leq \text{data}(y_2))
    \]
Our Contribution

- A new logic, called STRAND
  - defined over a class of recursive data structure $\mathcal{R}$
  - $\exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y})$
  - where $\varphi$ is an Monadic Second Order (MSO) formula
    combines heap structures and data, but where the data-constraints are only allowed to refer to $\vec{x}$ and $\vec{y}$
  - Example:
    $\forall y_1 \forall y_2. (\exists z. (y_1 \rightarrow z \land z \rightarrow y_2) \Rightarrow (data(y_1) \leq data(y_2)))$

- Identify a decidable fragment of STRAND
  - semantically defined, but syntactically checkable
  - based on the notion of satisfiability-preserving embeddings
Our Contribution

- A new logic, called STRAND
  - defined over a class of recursive data structure $R$
  - $\exists \bar{x} \forall \bar{y} \varphi(\bar{x}, \bar{y})$
  - where $\varphi$ is an Monadic Second Order (MSO) formula combines heap structures and data, but where the data-constraints are only allowed to refer to $\bar{x}$ and $\bar{y}$
  - Example:
    $\forall y_1 \forall y_2. (\exists z. (y_1 \rightarrow z \land z \rightarrow y_2) \Rightarrow (data(y_1) \leq data(y_2)))$

- Identify a decidable fragment of STRAND
  - semantically defined, but syntactically checkable
  - based on the notion of satisfiability-preserving embeddings

- On certain classes of recursive data structures, identify a decidable syntactic fragment of STRAND
Combining Theories

Traditional:
- Nelson-Oppen approach
- two-way connection, quantifier free
- full FOL is undecidable

Our Scheme:
Contract a Sorted List

\[ \varphi_1 : \quad d(\text{head})=1 \land d(\text{tail})=10^6 \land \\
\forall y_1 \forall y_2 . ((y_1 \rightarrow y_2) \Rightarrow d(y_1) \leq d(y_2)) \]

\[ \widehat{\varphi}_1 : \quad p_1(\text{head}) \land p_2(\text{tail}) \land \\
\forall y_1 \forall y_2 . ((y_1 \rightarrow y_2) \Rightarrow p_3(y_1, y_2)) \]

(for any \( y_1, y_2, p_1, p_2, p_3 \) hold for free!)
One Million Example

\[ \varphi_2 : \ d(\text{head})=1 \ \land \ d(\text{tail})=10^6 \ \land \ \\
\forall y_1 \forall y_2 . ((y_1 \rightarrow y_2) \Rightarrow d(y_2) = d(y_1) + 1) \]

\[ \tilde{\varphi}_2 : \ p_1(\text{head}) \ \land \ p_2(\text{tail}) \ \land \ \\
\forall y_1 \forall y_2 . ((y_1 \rightarrow y_2) \Rightarrow p_3(y_1, y_2)) \]

(When \( y_1 = \text{head} \), \( y_2 = \text{tail} \), \( p_3(\text{head}, \text{tail}) \) does not hold for free!)
Outline

Overview

**STRAND Logic**

Deciding STRAND Fragments

Program Verification using STRAND
Recursive data-structures

Example

leaves of the tree are connected by a linked list.
Recursive data-structures

Example

\[ E_{next}(s, t) \equiv \text{leaf}(s) \land \text{leaf}(t) \land \exists z_1, z_2, z_3 (E_l(z_3, z_1) \land E_r(z_3, z_2) \land \text{RightMostPath}(z_1, s) \land \text{LeftMostPath}(z_2, t)) \]
Syntax of STRAND Logic

$$\exists \text{DVar} \quad x \in Loc$$
$$\forall \text{DVar} \quad y \in Loc$$
$$\text{GVar} \quad z \in Loc$$

Variable $$v ::= x \mid y \mid z$$

Set – Variable $$S \in 2^{Loc}$$

Constant $$c \in \text{Sig}(\mathcal{D})$$

Function $$g \in \text{Sig}(\mathcal{D})$$

$$\mathcal{D}$$–Relation $$\gamma \in \text{Sig}(\mathcal{D})$$

$$\mathcal{L}$$–Relation $$\alpha \in \text{Sig}(\mathcal{L})$$

Expression $$e ::= \text{data}(x) \mid \text{data}(y) \mid c \mid g(e_1, \ldots, e_n)$$

AFormula $$\varphi ::= \gamma(e_1, \ldots, e_n) \mid \alpha(v_1, \ldots, v_n)$$
$$\mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2$$
$$\mid \exists z. \varphi \mid \forall z. \varphi \mid \exists S. \varphi \mid \forall S. \varphi$$

$$\exists$$Formula $$\omega ::= \varphi \mid \forall y. \omega$$

Formula $$\psi ::= \omega \mid \exists x. \psi$$
Example: Binary Search Tree

\[ \text{leftbranch}(y_1, y_2) \equiv \exists z (\text{left}(y_1, z) \land z \rightarrow^* y_2) \]

\[ \text{rightbranch}(y_1, y_2) \equiv \exists z (\text{right}(y_1, z) \land z \rightarrow^* y_2) \]

\[ \psi_{bst} \equiv \forall y_1 \forall y_2 ( (\text{leftbranch}(y_1, y_2) \Rightarrow d(y_2) < d(y_1)) \land \]

\[ ((\text{rightbranch}(y_1, y_2) \Rightarrow d(y_1) \leq d(y_2)) ) \]
Example: Two disjoint lists

(head₁ →* tail₁) * (head₂ →* tail₂) states, in separation logic, that there are two disjoint lists such that one list is from head₁ to tail₁, and the other is from head₂ to tail₂.

ψ₂lists ≡ ∃S₁ ∃S₂(disjoint(S₁, S₂) ∧

head₁ ∈ S₁ ∧ tail₁ ∈ S₁ ∧ head₂ ∈ S₂ ∧ tail₂ ∈ S₂ ∧

∀z.((head₁ →* z ∧ z →* tail₁) ⇒ z ∈ S₁) ∧

∀z.((head₂ →* z ∧ z →* tail₂) ⇒ z ∈ S₂) ∧

head₁ →* tail₁ ∧ head₂ →* tail₂)

where disjoint(S₁, S₂) is defined as

¬∃z(z ∈ S₁ ∧ z ∈ S₂)
Outline

Overview

STRAND Logic

Deciding STRAND Fragments

Program Verification using STRAND
Satisfiability-Preserving Embeddings

- $\exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y})$ over $R$ can be transformed to an equisatisfiable formula $\forall \vec{x} \forall \vec{y} \varphi'(\vec{x}, \vec{y})$ over $R'$
Satisfiability-Preserving Embeddings

- $\exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y})$ over $\mathcal{R}$ can be transformed to an equisatisfiable formula $\forall \vec{x} \forall \vec{y} \varphi'(\vec{x}, \vec{y})$ over $\mathcal{R}'$

- Question: How to define the satisfiability-preserving embeddings?
Satisfiability-Preserving Embeddings

- \( \exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y}) \) over \( \mathcal{R} \) can be transformed to an equisatisfiable formula \( \forall \vec{x} \forall \vec{y} \varphi'(\vec{x}, \vec{y}) \) over \( \mathcal{R}' \)

- Question: How to define the satisfiability-preserving embeddings?

- Needed: if the larger graph model can satisfy \( \varphi \) with some data extension, the smaller model can also satisfy \( \varphi \) with some data extension.
Intuition

- since the data-values in the submodel are inherited from the larger model, the atomic data-relations would hold in the same way as they do in the larger model

- \( S \) satisfiability-preservingly embeds in \( T \) iff no matter how \( T \) satisfies the formula using some valuation of the atomic data-relations, \( S \) will be able to satisfy the formula using the same valuation of the atomic data-relations
Observation

\textbf{STRAND}_{dec}: formulas that have a \textit{finite number of minimal models} w.r.t the partial-order defined by satisfiability-preserving embeddings.
**Observation**

**STRAND\_\text{dec}:** formulas that have a finite number of minimal models w.r.t the partial-order defined by satisfiability-preserving embeddings.

Question: Is $\psi = \forall \vec{y} \varphi(\vec{y}) \in \text{STRAND}\_\text{dec}$?

(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the atomic relational formulas of the data-logic in $\varphi$)
**Observation**

**STRAND\textsubscript{dec}:** formulas that have a finite number of minimal models w.r.t the partial-order defined by satisfiability-preserving embeddings.

Question: Is $\psi = \forall \vec{y} \varphi(\vec{y}) \in \text{STRAND}\textsubscript{dec}$?

(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the atomic relational formulas of the data-logic in $\varphi$)

Observation 1: After fixing a particular valuation of $\vec{y}$, all data-relations $\gamma_i$ get all fixed.
Observation

**STRAND}_{dec}:** formulas that have a **finite number of minimal models** w.r.t the partial-order defined by satisfiability-preserving embeddings.

Question: Is $\psi = \forall \vec{y} \varphi(\vec{y}) \in \text{STRAND}_{dec}$?

(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the **atomic** relational formulas of the data-logic in $\varphi$)

Observation 1: After fixing a particular valuation of $\vec{y}$, all data-relations $\gamma_i$ get all fixed

Observation 2: No matter how we choose to evaluate $\vec{y}$ over the nodes of the model, the $\gamma_i$ relations must evaluate to true or false in such a way that $\varphi$ holds
Minimal Model

Solution: From $\varphi(\vec{y})$, abstract $\gamma_i$ as a predicate $p_i$ to get a pure structural formula $\widehat{\varphi}(\vec{y}, \vec{p})$!
Minimal Model

Solution: From $\varphi(\vec{y})$, abstract $\gamma_i$ as a predicate $p_i$ to get a pure structural formula $\hat{\varphi}(\vec{y}, \vec{p})$!

$$MinModel = \neg \exists X. (\text{ValidSubModel}(X) \land$$

$$\forall \vec{y} \forall \vec{p} ((\land_{y \in \vec{y}} (y \in X) \land \varphi(\vec{y}, \vec{p}))$$

$$\Rightarrow \text{tailor}_X(\hat{\varphi}(\vec{y}, \vec{p}))))$$

$\text{tailor}_X$: transform $\hat{\varphi}$ to a formula that expresses the same property on the submodel defined $X$. 
Decision Procedure

- define the MSO formula on $k$-ary trees $MinModel$ that captures minimal models.

- transform the MSO formula to a tree automaton that accepts precisely those trees that satisfy the formula.

- Since the finite-ness of the language accepted by a tree automaton is decidable, $\text{STRAND}_{dec}$ is effectively checkable!
A Decidable Syntactic Fragment of STRAND

$R$: Trees only allow $\text{LeftSubtree}$ and $\text{RightSubtree}$

Syntax: $\exists \vec{x} \forall \vec{y} \; \varphi(\vec{x}, \vec{y})$ where $\varphi$ is quantifier-free

(for any $y_1, y_2$, they are related in $T$ iff they are related in $\text{Submodel}(T, S)$, hence decidable!)
Outline

Overview

STRAND Logic

Deciding STRAND Fragments

Program Verification using STRAND
Program Verification Overview

Hoare-triples: \((R, \text{Pre}, P, \text{Post})\)

\(R\)

\(\text{Pre} \in \text{Strand}_{\exists, \forall}\)

\(P:\)

\(\text{Node } t = \text{newhead};\)
\(\text{newhead} = \text{head};\)
\(\text{head} = \text{head} . \text{next};\)
\(\text{newhead} . \text{next} = t;\)

\(\text{Post} \in \text{Strand}_{\exists, \forall}\)

\(\implies \text{Is } \psi \in \text{Strand}\)

Satisfiable over \(R_P\)?

Idea: capture the entire computation \(P\) starting from a particular recursive data-structure \(R\) using a single data-structure \(R_P\)
The Hoare-triple $(\mathcal{R}, \text{Pre}, \text{P}, \text{Post})$ does not hold iff the STRAND formula $\text{Error} \lor \text{Violate}_{\text{Post}}$ is satisfiable on the trail $\mathcal{R}_P$. 

\[ \text{Error} = \bigvee_{i \in [m]} (\text{Pre}_{\mathcal{R}_P} \land \bigwedge_{j \in [i-1]} \varphi_j \land \text{error}_i) \]

\[ \text{Violate}_{\text{Post}} = \text{Pre}_{\mathcal{R}_P} \land (\bigwedge_{j \in [m]} \varphi_j) \land \lnot \text{Post}_{\mathcal{R}_P} \]
bstSearch

(pre: $\psi_{bst} \land \exists x. (\text{key}(\text{root}) = k)$)

Node curr = root;
(loop-inv: $\psi_{bst} \land \exists x. (\text{reach}(\text{curr}, x) \land \text{key}(\text{curr}) = k)$)
while (curr.key != k / curr != nil){
  if (curr.key > k) curr = curr.left;
  else curr = curr.right;
}
(post: $\psi_{bst} \land \text{key}(\text{curr}) = k$)
macro minimalmodel(var2 $, var1 curr, var1 curr1, var1 exdv1, var1 exdv2, var1 anotherk, var0 pc1, var0 pc2, var0 pc3, var0 pc4, var0 pc12, var0 pc22) =

  ~ex2 M where M sub $ & M^\gamma=$:(
    validmodel'($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,M) &
    all1 dv1,dv2,dv3: ( (dv1 in M & dv2 in M & dv3 in M) =>
      (all0 p11,p21,p3: ( ((~precondition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | precondition'($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3,M)) | ~negpostcondition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3)) &
        (~precondition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | (~negpostcondition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | negpostcondition'($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3,M)))
      )))};
## Results of Program Verification

<table>
<thead>
<tr>
<th>Program</th>
<th>Verification condition</th>
<th>Structural solving (MONA)</th>
<th>Data-constraint Solving (Z3 with QF-LIA)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>in STRAND&lt;sub&gt;dec&lt;/sub&gt;? (finitely-many minimal models)</td>
<td>Time(s)</td>
</tr>
<tr>
<td>sorted-list-search</td>
<td>before-loop</td>
<td>Yes</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>before-head</td>
<td>Yes</td>
<td>1.66</td>
</tr>
<tr>
<td></td>
<td>before-loop</td>
<td>Yes</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>4.46</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>13.93</td>
</tr>
<tr>
<td>sorted-list-insert</td>
<td>before-loop</td>
<td>Yes</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>4.41</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>13.63</td>
</tr>
<tr>
<td>sorted-list-insert-error</td>
<td>before-loop</td>
<td>Yes</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>2.79</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>0.35</td>
</tr>
<tr>
<td>sorted-list-reverse</td>
<td>before-loop</td>
<td>Yes</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>2.79</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>0.35</td>
</tr>
<tr>
<td>bst-search</td>
<td>before-loop</td>
<td>Yes</td>
<td>5.03</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>32.80</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>3.27</td>
</tr>
<tr>
<td>bst-insert</td>
<td>before-loop</td>
<td>Yes</td>
<td>1.34</td>
</tr>
<tr>
<td></td>
<td>in-loop</td>
<td>Yes</td>
<td>9.84</td>
</tr>
<tr>
<td></td>
<td>after-loop</td>
<td>Yes</td>
<td>1.76</td>
</tr>
</tbody>
</table>

http://cs.uiuc.edu/~qiu2/strand/
Future Work

- Powerful and decidable syntactic fragments
- Back-and-force connection between the structural part and the data part
- Separation logic
<table>
<thead>
<tr>
<th>Overview</th>
<th>STRAND Logic</th>
<th>Deciding STRAND Fragments</th>
<th>Program Verification using STRAND</th>
</tr>
</thead>
</table>

Questions?