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

Logic and SMT Solvers

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

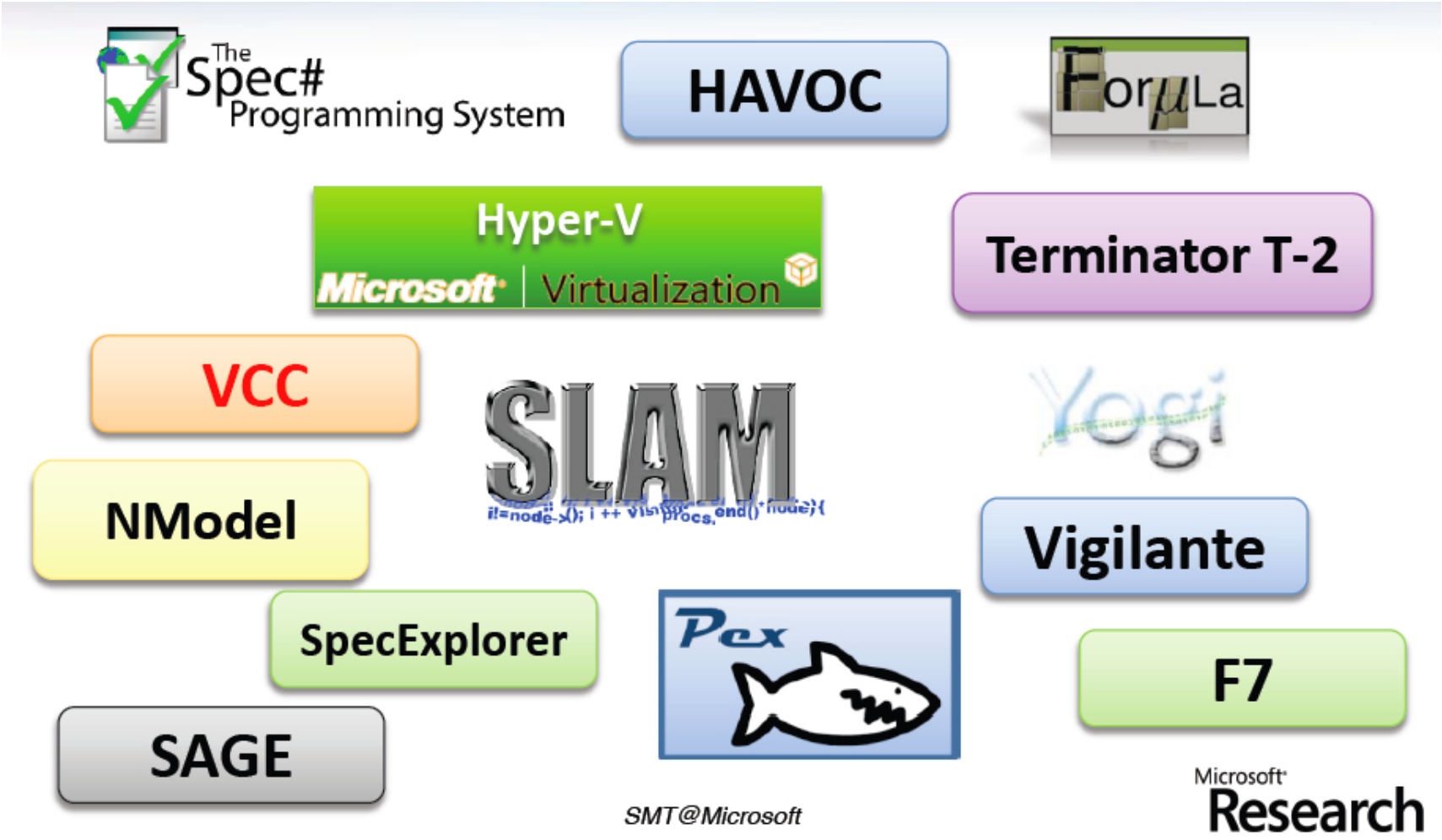
Logic and SMT Solvers

- 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 φ 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:
    $\forall y_1 \forall y_2.(\exists z.(y_1 \rightarrow z \wedge z \rightarrow y_2) \Rightarrow (data(y_1) \leq data(y_2))$

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 φ 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:
    $\forall y_1 \forall y_2.(\exists z.(y_1 \rightarrow z \wedge z \rightarrow y_2) \Rightarrow (data(y_1) \leq data(y_2))$

- 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 $\mathcal{R}$
  - $\exists \vec{x}\ \forall \vec{y}\ \varphi(\vec{x}, \vec{y})$
  - where φ 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:
    $\forall y_1 \forall y_2.(\exists z.(y_1 \rightarrow z \wedge z \rightarrow y_2) \Rightarrow (data(y_1) \leq data(y_2))$

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

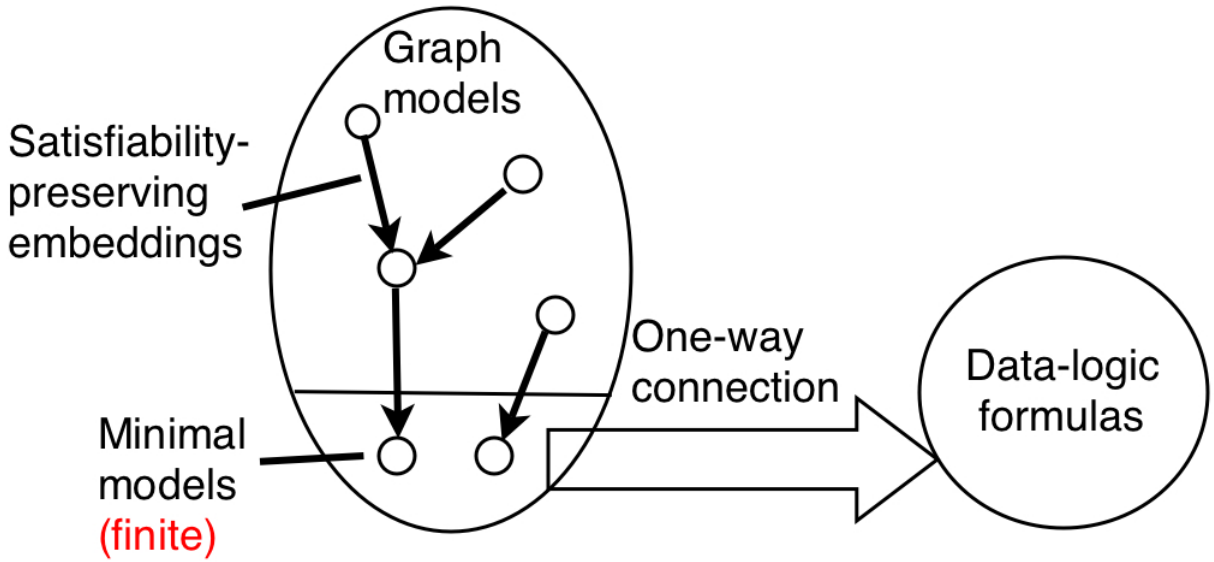
- On certain classes of recursive data structures, identify a decidable syntactic fragment of STRAND

Combining Theories

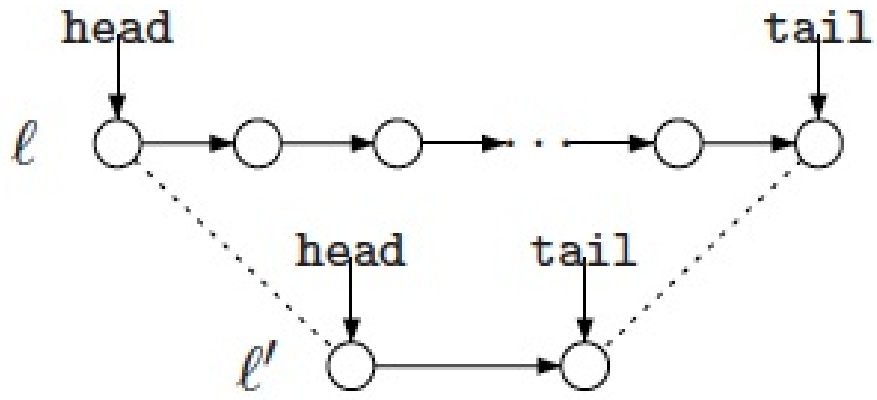
Traditional:

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

Our Scheme:



Contract a Sorted List

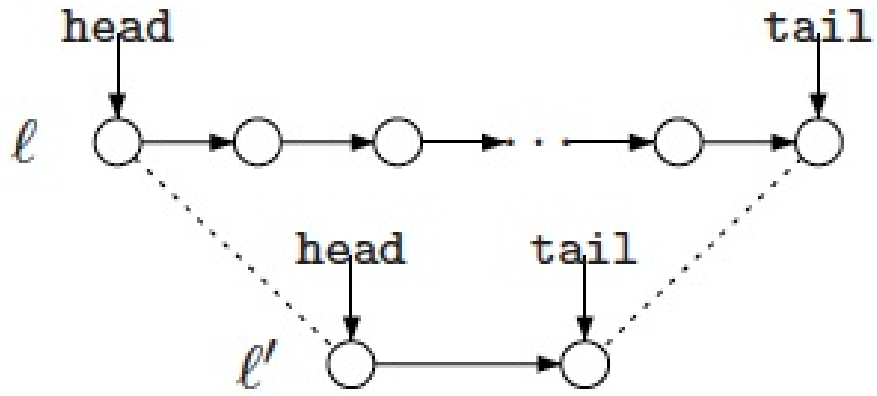


$$\varphi_1 : \quad d(\texttt{head})=1 \ \wedge \ d(\texttt{tail})=10^6 \ \wedge \ \forall y_1 \forall y_2.((y_1 \rightarrow^* y_2) \Rightarrow d(y_1) \leq d(y_2))$$

$$\widehat{\varphi_1} : \quad p_1(\texttt{head}) \ \wedge p_2(\texttt{tail}) \ \wedge \ \forall y_1 \forall y_2.((y_1 \rightarrow^* y_2) \Rightarrow p_3(y_1, y_2))$$

(for any y_1, y_2, p_1, p_2, p_3 hold for free!)

One Million Example



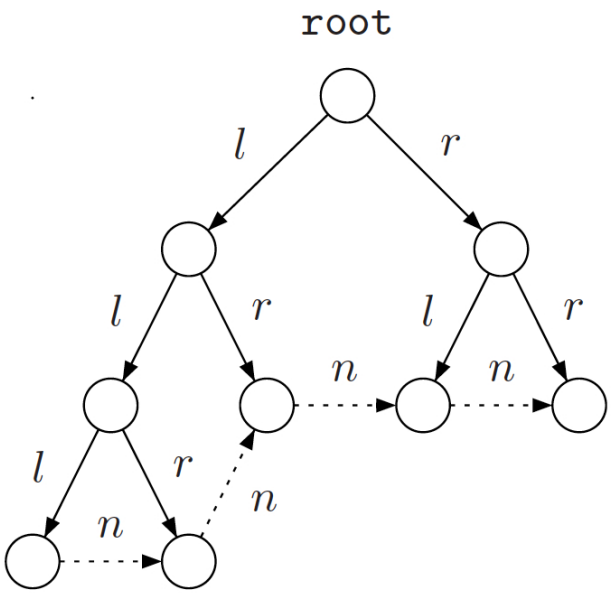
$$\varphi_2 : \quad d(\texttt{head})=1 \ \wedge \ d(\texttt{tail})=10^6 \ \wedge \ \forall y_1 \forall y_2.((y_1 \rightarrow y_2) \Rightarrow d(y_2) = d(y_1) + 1))$$

$$\widehat{\varphi_2} : \quad p_1(\texttt{head}) \ \wedge \ p_2(\texttt{tail}) \ \wedge \ \forall y_1 \forall y_2.((y_1 \rightarrow y_2) \Rightarrow p_3(y_1, y_2)))$$

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

Outline

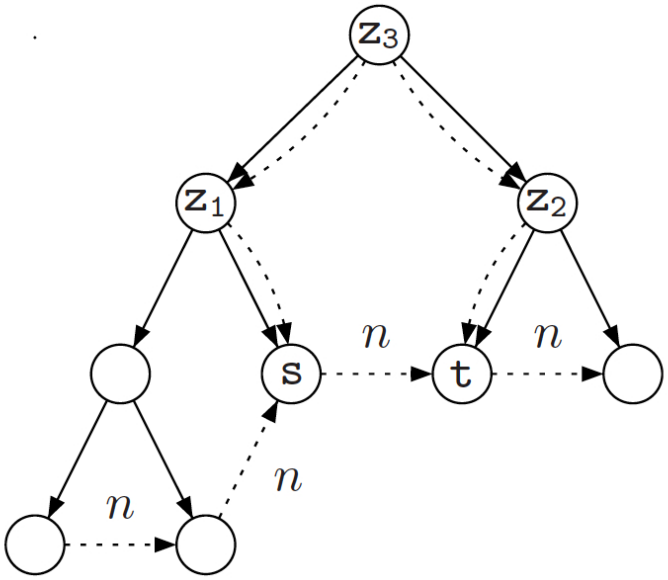
Recursive data-structures



Example

leaves of the tree are connected by a linked list.

Recursive data-structures



Example

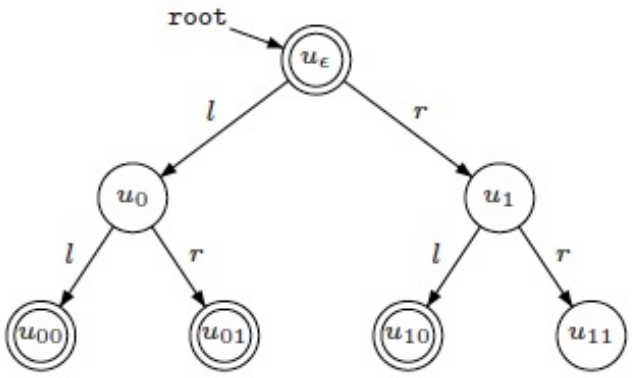
$$E_{next}(s,t) \equiv leaf(s) \wedge leaf(t) \wedge \exists z_1, z_2, z_3 (E_l(z_3, z_1) \wedge E_r(z_3, z_2) \wedge RightMostPath(z_1, s) \wedge LeftMostPath(z_2, t))$$

Syntax of STRAND Logic

$\exists$DVar	x	$\in$	Loc
$\forall$DVar	y	$\in$	Loc
GVar	z	$\in$	Loc
Variable	v	$::=$	$x \mid y \mid z$
Set – Variable	S	$\in$	2^{Loc}
Constant	c	$\in$	$Sig(\mathcal{D})$
Function	g	$\in$	$Sig(\mathcal{D})$
$\mathcal{D}$–Relation	γ	$\in$	$Sig(\mathcal{D})$
$\mathcal{L}$–Relation	α	$\in$	$Sig(\mathcal{L})$

Expression	e	$::=$	$data(x) \mid data(y) \mid c \mid g(e_1, \ldots, e_n)$
AFormula	φ	$::=$	$\gamma(e_1, \ldots, e_n) \mid \alpha(v_1, \ldots, v_n)$
			$\mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$
			$\mid \exists z.\varphi \mid \forall z.\varphi \mid \exists S.\varphi \mid \forall S.\varphi$
$\forall$Formula	ω	$::=$	$\varphi \mid \forall y.\omega$
Formula	ψ	$::=$	$\omega \mid \exists x.\psi$

Example: Binary Search Tree



$$leftbranch(y_1, y_2) \equiv \exists z(left(y_1, z) \wedge z \rightarrow^* y_2)$$

$$rightbranch(y_1, y_2) \equiv \exists z(right(y_1, z) \wedge z \rightarrow^* y_2)$$

$$\psi_{bst} \equiv \forall y_1 \forall y_2(\ (leftbranch(y_1, y_2) \Rightarrow d(y_2) < d(y_1)) \wedge$$

$$((rightbranch(y_1, y_2) \Rightarrow d(y_1) \leq d(y_2))\ )$$

Example: Two disjoint lists

$(\mathtt{head}_1 \rightarrow^* \mathtt{tail}_1) * (\mathtt{head}_2 \rightarrow^* \mathtt{tail}_2)$ states, in separation logic, that there are two disjoint lists such that one list is from $\mathtt{head}_1$ to $\mathtt{tail}_1$, and the other is from $\mathtt{head}_2$ to $\mathtt{tail}_2$.

$$\psi_{2lists} \equiv \exists S_1 \exists S_2 (\mathit{disjoint}(S_1, S_2) \wedge$$

$$\mathtt{head}_1 \in S_1 \wedge \mathtt{tail}_1 \in S_1 \wedge \mathtt{head}_2 \in S_2 \wedge \mathtt{tail}_2 \in S_2 \wedge$$

$$\forall z.((\mathtt{head}_1 \rightarrow^* z \wedge z \rightarrow^* \mathtt{tail}_1) \Rightarrow z \in S_1) \wedge$$

$$\forall z.((\mathtt{head}_2 \rightarrow^* z \wedge z \rightarrow^* \mathtt{tail}_2) \Rightarrow z \in S_2) \wedge$$

$$\mathtt{head}_1 \rightarrow^* \mathtt{tail}_1 \wedge \mathtt{head}_2 \rightarrow^* \mathtt{tail}_2)$$

where $\mathit{disjoint}(S_1, S_2)$ is defined as

$$\nexists z (z \in S_1 \wedge z \in S_2)$$

Outline

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 $\mathcal{R}'$

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?

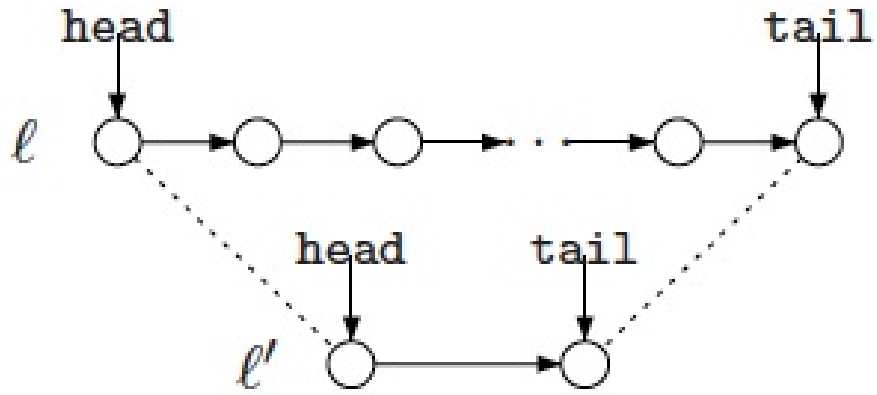
Satisfiability-Preserving Embeddings

- $\exists \vec{x}\ \forall \vec{y}\ \varphi(\vec{x}, \vec{y})$ 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 φ with some data extension, the smaller model can also satisfy φ 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

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

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 $\psi = \forall \vec{y}\, \varphi(\vec{y}) \in$ STRAND$_{dec}$?
(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the atomic relational formulas of the data-logic in φ)

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 $\psi = \forall \vec{y}\ \varphi(\vec{y}) \in$ STRAND$_{dec}$?
(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the atomic relational formulas of the data-logic in φ)

Observation 1: After fixing a particular valuation of $\vec{y}$, all data-relations γ_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 $\psi = \forall \vec{y}\ \varphi(\vec{y}) \in$ STRAND$_{dec}$?
(Let $\gamma_1, \gamma_2, \ldots, \gamma_r$ be the atomic relational formulas of the data-logic in φ)

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

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

Minimal Model

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

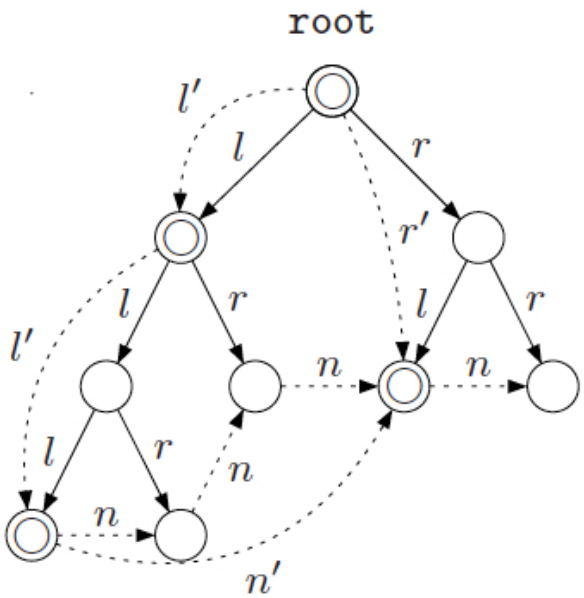
$$\begin{aligned} \mathit{MinModel} = \neg\exists X.(\, & \mathit{ValidSubModel}(X) \wedge \\ & (\ \forall \vec{y}\ \forall \vec{p}\ ((\wedge_{y \in \vec{y}}(y \in X) \wedge \widehat{\varphi}(\vec{y}, \vec{p})) \\ & \Rightarrow \mathit{tailor}_X(\widehat{\varphi}(\vec{y}, \vec{p})))))\end{aligned}$$

tailor_X: transform $\widehat{\varphi}$ 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, STRAND_{dec} is effectively checkable!

A Decidable Syntactic Fragment of STRAND



$\mathcal{R}$: Trees only allow *LeftSubtree* and *Rightsubtree*

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

(for any y_1, y_2, they are related in T iff they are related in *Submodel*(T, S), hence decidable!)

Outline

Program Verification Overview

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

$\mathcal{R}$
$Pre \in Strand_{\exists,\forall}$
P :

```
Node t = newhead;
newhead = head;
head = head.next;
newhead.next = t;
```

$Post \in Strand_{\exists,\forall}$

$\Rightarrow$ Is $\psi \in Strand$
Satisfiable over $\mathcal{R}_P$?

Idea: capture the entire computation P starting from a particular recursive data-structure $\mathcal{R}$ using a single data-structure $\mathcal{R}_P$

result

$$Error = \bigvee_{i \in [m]} (Pre_{\mathcal{R}_P} \wedge \bigwedge_{j \in [i-1]} \varphi_j \quad \wedge \quad error_i)$$

$$Violate_{Post} = Pre_{\mathcal{R}_P} \wedge (\bigwedge_{j \in [m]} \varphi_j) \wedge \neg Post_{\mathcal{R}_P}$$

Theorem

The Hoare-triple $(\mathcal{R}, Pre, P, Post)$ *does not hold iff the* STRAND *formula* $Error \vee Violate_{Post}$ *is satisfiable on the trail* $\mathcal{R}_P$.

bst-search-in-loop

```
bstSearch
  (pre: ψ_bst ∧ ∃x.(key(root) = k))
  Node curr = root;
  (loop-inv: ψ_bst ∧ ∃x.(reach(curr, x) ∧ key(curr) = k))
  while (curr.key != k / curr != nil){
    if (curr.key > k) curr = curr.left;
    else curr = curr.right;
  }
  (post: ψ_bst ∧ key(curr) = k)
```

Formula

```
macro minimalmodel(var2 $, var1 curr, var1 curr1, var1 exdv1, var1 exdv2, var1 a
notherk, var0 pc1, var0 pc2, var0 pc3, var0 pc4, var0 pc12, var0 pc22) =
        ~ex2 M where M sub $ & M~=$:(
        validmodel'($,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($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc
2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | precondition'($,curr,curr1,exdv1,e
xdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3,M)) | ~negpostcon
dition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p
11,p21,p3)) &
                        (~precondition($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2
,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | (~negpostcondition($,curr,curr1,exd
v1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,p11,p21,p3) | negpostcon
dition'($,curr,curr1,exdv1,exdv2,anotherk,pc1,pc2,pc3,pc4,pc12,pc22,dv1,dv2,dv3,
p11,p21,p3,M)))
                ))));
```

Results of Program Verification

Program	Verification condition	Structural solving (MONA)			Data-constraint Solving (Z3 with QF-LIA)		
		in STRAND$_{dec}$? (finitely-many minimal models)	Time(s)	Graph model exists?	Bound (#Nodes)	Satisfiable?	Time(s)
sorted-list-search	before-loop	Yes	0.34	No	-	-	-
	in-loop	Yes	0.59	No	-	-	-
	after-loop	Yes	0.18	No	-	-	-
sorted-list-insert	before-head	Yes	1.66	Yes	5	No	0.02
	before-loop	Yes	0.38	No	-	-	-
	in-loop	Yes	4.46	No	-	-	-
	after-loop	Yes	13.93	Yes	7	No	0.02
sorted-list-insert-error	before-loop	Yes	0.34	Yes	7	Yes	0.02
	in-loop	Yes	4.41	No	-	-	-
	after-loop	Yes	13.63	Yes	7	No	0.02
sorted-list-reverse	before-loop	Yes	0.24	No	-	-	-
	in-loop	Yes	2.79	No	-	-	-
	after-loop	Yes	0.35	No	-	-	-
bst-search	before-loop	Yes	5.03	No	-	-	-
	in-loop	Yes	32.80	Yes	9	No	0.02
	after-loop	Yes	3.27	No	-	-	-
bst-insert	before-loop	Yes	1.34	No	-	-	-
	in-loop	Yes	9.84	No	-	-	-
	after-loop	Yes	1.76	No	-	-	-

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

Questions?