

Decidable Logics Combining Heap Structures and Data

Xiaokang Qiu¹

P. Madhusudan¹ Gennaro Parlato²

¹University of Illinois at Urbana-Champaign

²LIAFA, CNRS and University of Paris

POPL'11(Principles of Programming Languages)

SMT Solvers: Engines of logical reasoning

SMT solvers

- check satisfiability in particular theories (usually QF)
- engines of proof that serve many programming verification and analysis techniques

Myriad applications

- Test input generation
- Verifying compilers
- Computing abstractions
- Invariant generation
- Floyd-Hoare style deductive verification

Theories Supported by SMT Solvers

- Equality over uninterpreted functions and predicate symbols
- Real and integer arithmetic
- Bit-vectors
- Arrays
- Tuple/Record/Enumeration types and algebraic data-types,...
- **Nelson-Oppen**: Combining Quantifier-free theories
- Heap structures + Data?

Motivation

Reasoning with Heap structures + Data

- E.g., Verification conditions for verifying binary-search tree search
If the key k is below $curr$, then in the next iteration of the loop, when $curr$ is updated, k will still be below $curr$.

Previous work

- HAVOC (Lahiri & Qadeer, POPL'08)
- CSL (Bouajjani et al., CONCUR'09)

Challenge

Unbounded size of heap + Quantification

Modeling Heaps

- Heaps = Graphs + Data
 - **heap locations:** nodes of the graph
 - **field pointer f from n to n' :** an edge from n to n' labeled f
 - **data field:** function $d : Loc \rightarrow Data$
- Decidable Logic for Heaps =
Combination of
 - A Decidable Logic on Graphs
 - +
 - Decidable Quantifier-free Data-logic (e.g. QF-integers)

Decidable Logic on Graphs

Monadic Second-Order (MSO) logic on **tree-like graphs**

Tree-like graphs:

- a skeleton tree
- precise set of nodes: unary predicate in MSO
- edges: binary relations in MSO

Quite powerful: All bounded tree-width graphs;
most recursive data-structures of interest

Recursive data-structures

A regular class of skeleton trees + unary predicates + binary relations

A class of graphs defined by

- **Regular class of trees:** A MSO formula ψ_{Tr}
- **Edge relations E_a :** given as an MSO formula $E_a(x, y)$
- **Vertex labels:** given as an MSO formula $L_b(x)$

DECIDABLE MSO (interpreting on trees; tree automata)

Similar to graph types (Klarlund & Schwartzbach, POPL'93)

Trees with all nodes pointing to root:

```
type Tree = {  
  data left, right: Tree;  
  pointer root: Tree[root < (left + right)* > this &  
    empty(root ^ Tree.left union  
    root ^ Tree.right)];  
}
```

Doubly-linked list:

```
type Node = {  
  bool value;  
  data next: Node;  
  pointer prev: Node[this ^ Node.next = {prev}];  
}
```

STRAND Logic

- A new logic *STRAND*

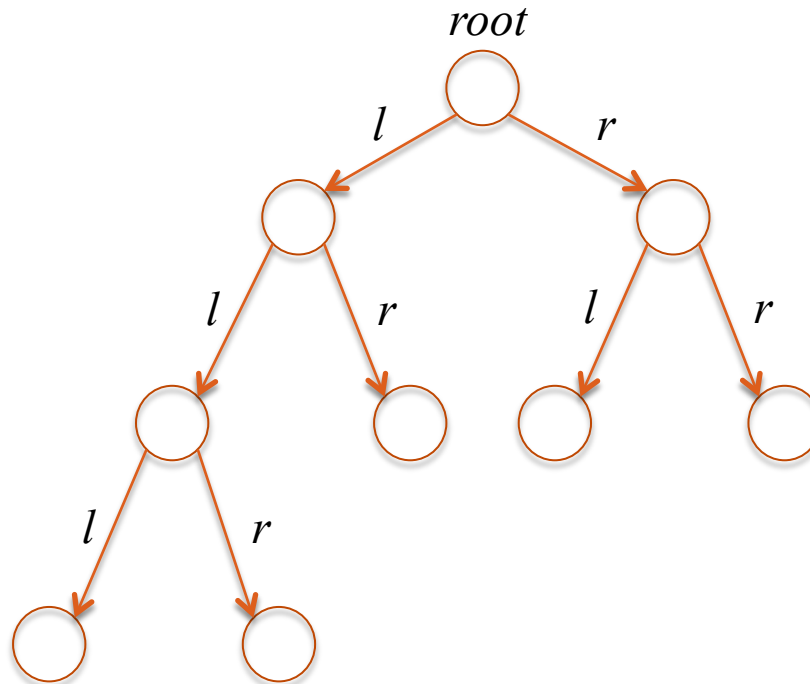
defined over a class of recursive data-structures $\mathcal{R}$

$$\exists \vec{x} \forall \vec{y} . \varphi(\vec{x}, \vec{y})$$

- where φ is a MSO formula enriched by data-field functions *data*, but where the data-constraints are only allowed to refer to $\vec{x}$ and $\vec{y}$.
- **Example:** A binary tree, with all data values at leaves being 1

$$\forall y. (\underbrace{\neg \exists z. (E_l(y, z) \wedge E_r(y, z))}_{\varphi} \Rightarrow (data(y) = 1))$$

Example: Binary Search Tree



$$\psi_{bst} \equiv \forall y_1, y_2. ((\text{leftsubtree}(y_1, y_2) \Rightarrow \text{data}(y_2) \leq \text{data}(y_1)) \wedge (\text{rightsubtree}(y_1, y_2) \Rightarrow \text{data}(y_2) > \text{data}(y_1))))$$

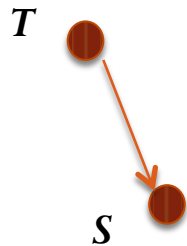
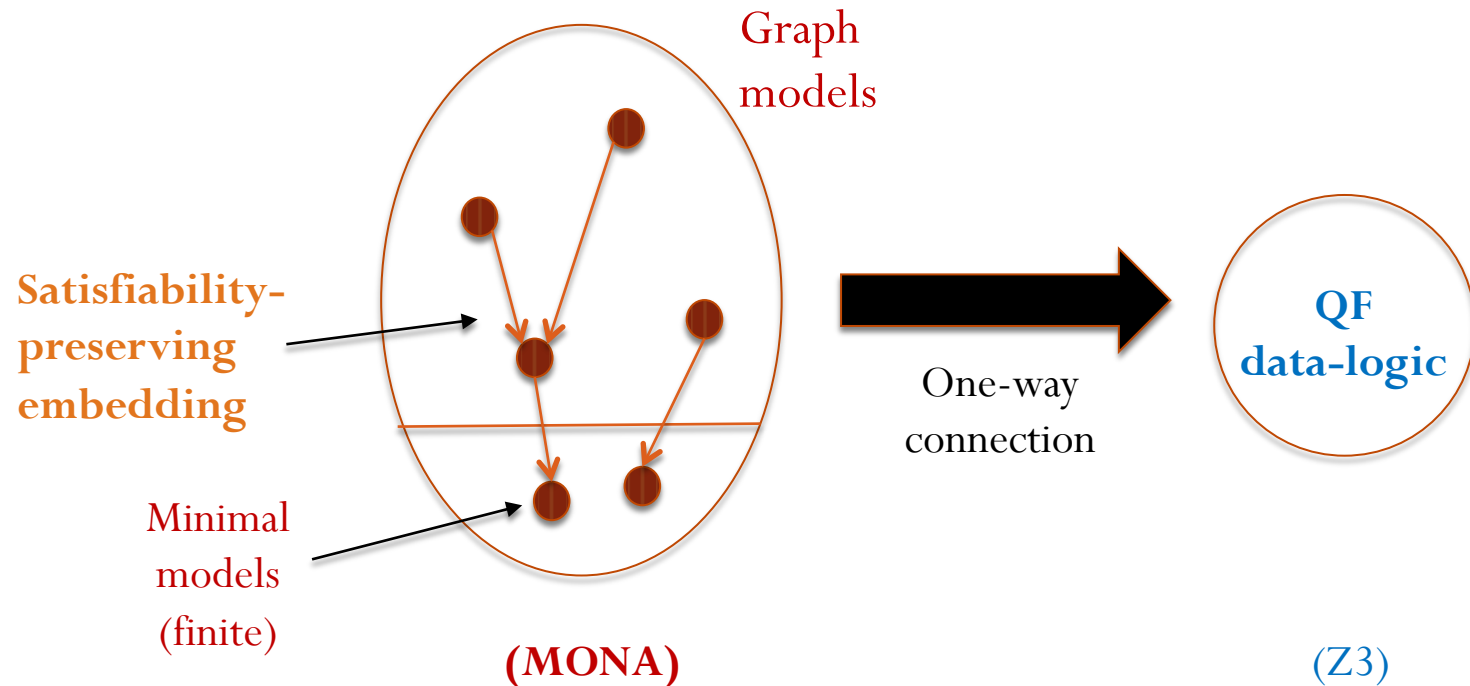
Decidable Fragments

- Identify a semantic decidable fragment $STRAND_{dec}^{sem}$
 - semantically defined, but syntactically checkable
- Identify a syntactic decidable fragment $STRAND_{dec}$
- Use minimal models according to a *data-agnostic* embedding relation

Outline

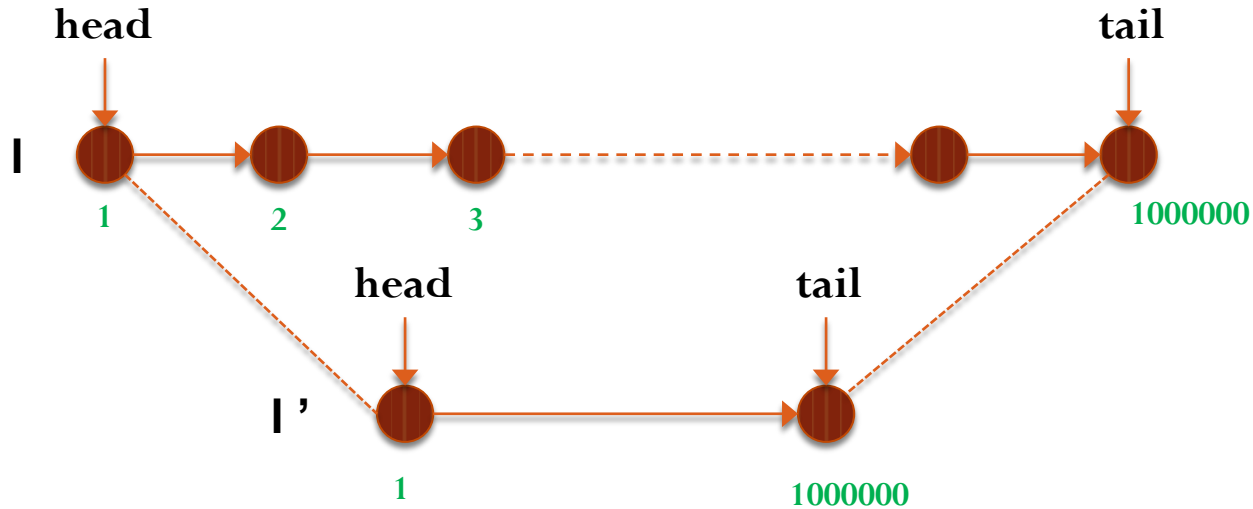
- Semantic Decidable Fragment
- Syntactic Decidable Fragment
- Program Verification
- Experimental Results

Our Scheme



If T has a data-extension that satisfies the STRAND formula,
then S also does.
Data-logic-agnostic!

Sorted List Embedding



$$\varphi_1 \equiv d(head) = 1 \wedge d(tail) = 10^6 \wedge \forall y_1 \forall y_2 ((y_1 \rightarrow^* y_2) \Rightarrow d(y_1) \leq d(y_2))$$

$$\hat{\varphi}_1 \equiv p_1(head) \wedge p_2(tail) \wedge \forall y_1 \forall y_2 ((y_1 \rightarrow^* y_2) \Rightarrow p_3(y_1, y_2))$$

Satisfiability-Preserving Embeddings

- 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
- T is a *minimal model* iff there is no proper submodel S of T such that S satisfiability-preservingly embeds in T
- $STRAND_{dec}^{sem}$: formulas that have finite number of minimal models w.r.t. the partial-order defined by satisfiability-preserving embeddings.

Observation

- **Consider:** $\psi = \exists \vec{x} \forall \vec{y}. \varphi(\vec{x}, \vec{y})$

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

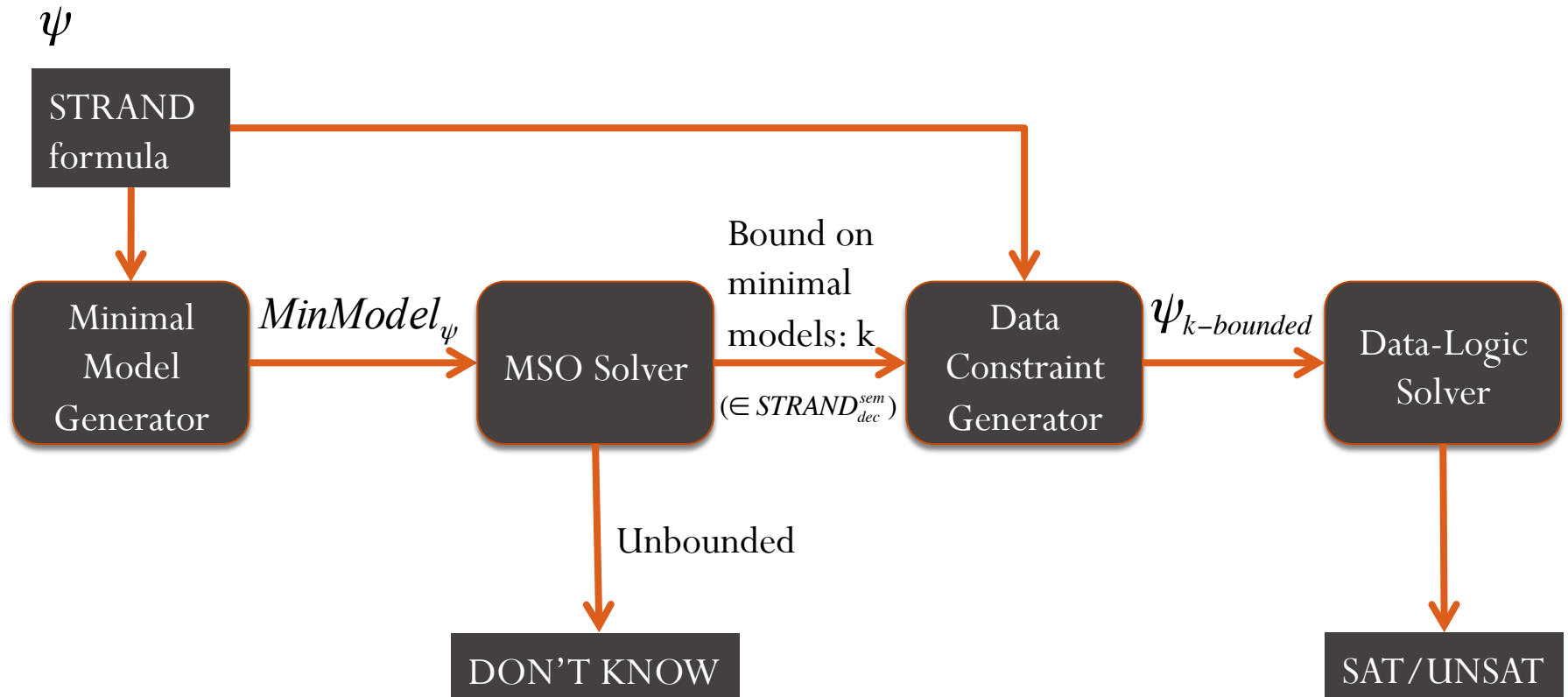
- Let M be a model of ψ , After fixing a particular valuation of $\vec{x}$ and $\vec{y}$, all data-relations γ_i get **all fixed to true or false**
- **Solution:** From $\varphi(\vec{x}, \vec{y})$, abstract γ_i as a predicate b_i to get a **pure structural** formula $\hat{\varphi}(\vec{x}, \vec{y}, \vec{b})$!

Minimal Model in MSO

$$\begin{aligned}
 \text{MinModel}_\psi \equiv & \exists \vec{x} . (\neg \exists S . (\\
 & \vec{x} \subseteq S \wedge \text{ValidSubmodel}(S) \wedge \\
 & \forall \vec{y} \in S, \vec{b} . (\\
 & \quad \hat{\varphi}(\vec{x}, \vec{y}, \vec{b}) \\
 & \quad \Rightarrow \text{Submodel}(S)[\hat{\varphi}(\vec{x}, \vec{y}, \vec{b})])) \\
 &) \\
 &)
 \end{aligned}$$

- transform the MSO formula to a tree automaton accepts precisely those minimal models
- Since the finite-ness of the language accepted by a tree automaton is decidable, $STRAND_{dec}^{sem}$ is effectively checkable!

Decision Procedure



Outline

- Semantic Decidable Fragment
- Syntactic Decidable Fragment
- Program Verification
- Experimental Results

A syntactic decidable fragment

$STRAND_{dec} : \exists \vec{x} \forall \vec{y}. \varphi(\vec{x}, \vec{y})$ where

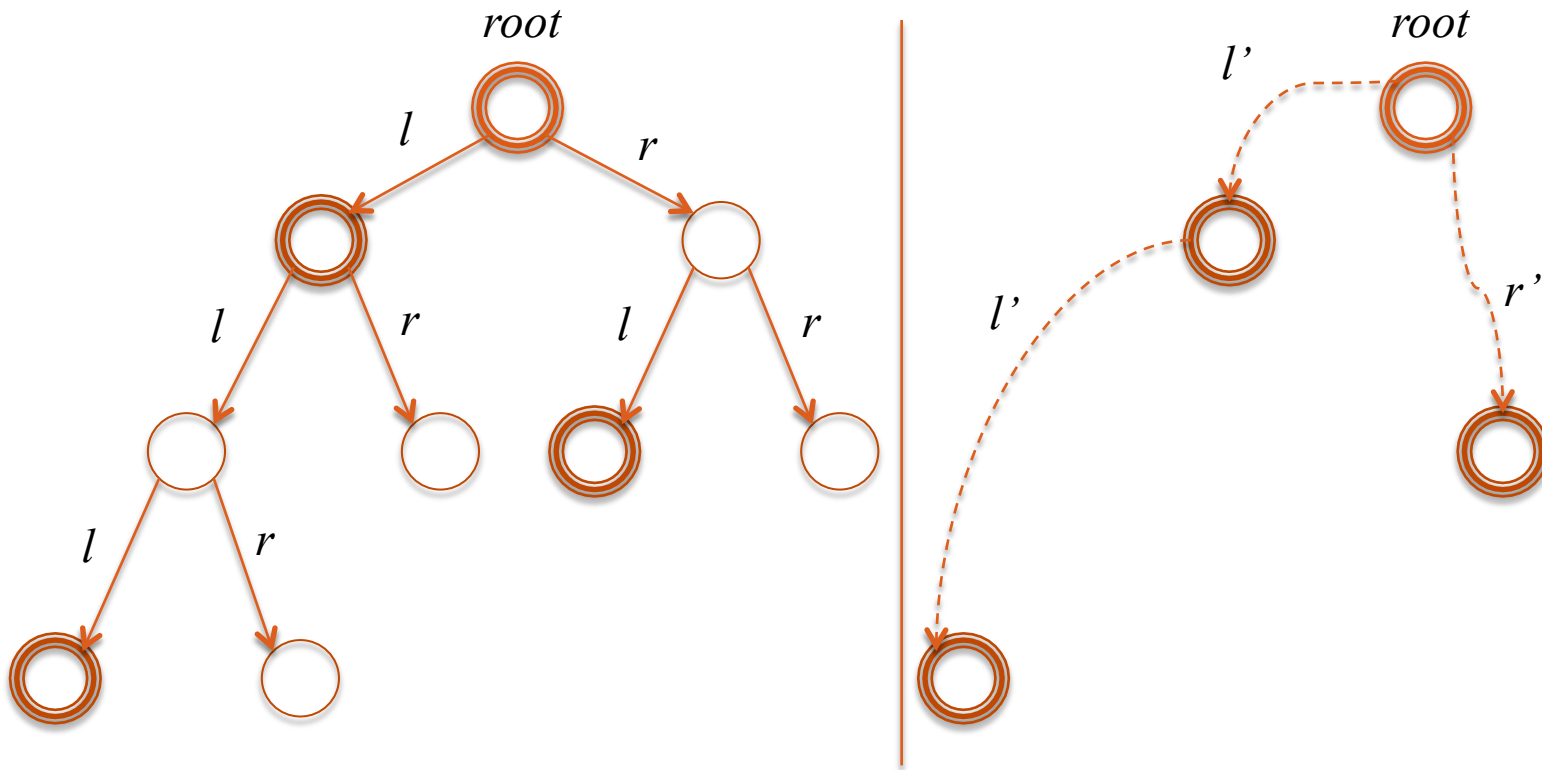
- φ has **no quantification**
- Elastic relations E and non-elastic relation NE
- Elastic relations are unconstrained
- Non-elastic relations only on $\vec{x}$ $NE(x_1, x_2)$

Elastic relations:

- $E(x, y)$ holds in model $\Leftrightarrow E(x, y)$ holds in any submodel

$STRAND_{dec}$ is **decidable**!

Elastic Relations



A relation R is **elastic** if for any model M and its submodel M' , for any $R(e_1, e_2)$, $e_1, e_2 \in M'$ holds for M iff $R(e_1, e_2)$ holds for M' .

E.g., ***LeftSubtree***(x, y) is elastic but ***LeftChild***(x, y) is not.

Outline

- Semantic Decidable Fragment
- Syntactic Decidable Fragment
- Program Verification
- Experimental Results

Verification Conditions

Hoare-triples: $(R, Pre, P, Post)$

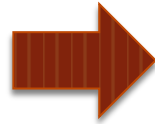
R : recursively defined data-structure

$Pre \in STRAND_{\exists, \forall}$

P :

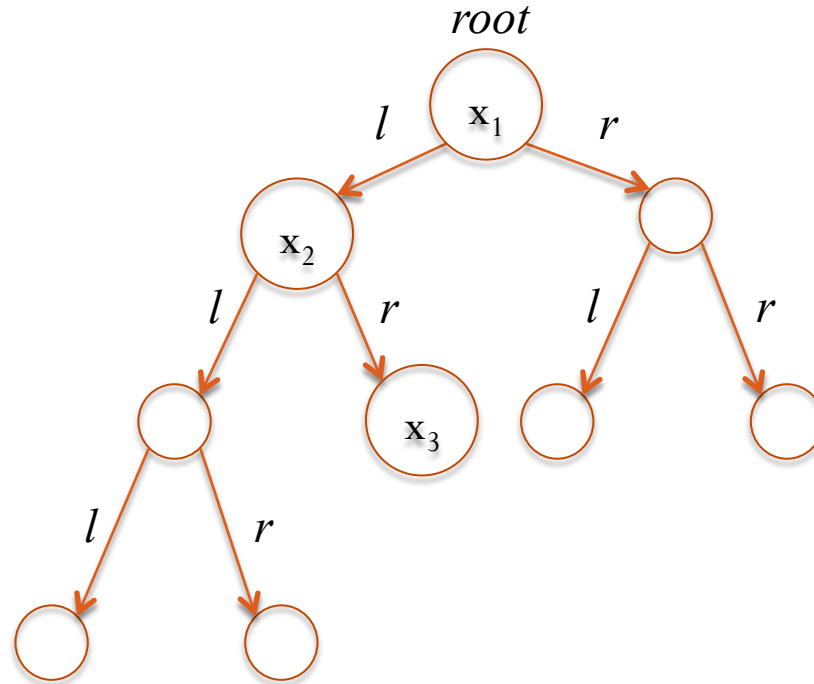
```
new newhead ;  
newhead.data := 14;  
newhead := head ;  
head := head.next ;  
newhead.next := t ;
```

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



Is $\psi \in STRAND$ satisfiable
over R_P ?

Idea: capture the entire computation P starting from a particular recursively defined data-structure R using a single data structure R_P



$$Error \equiv \bigvee_{i \in [m]} \left(Pre_{R_P} \wedge (\bigwedge_{j \in [i-1]} \varphi_j) \wedge error_i \right)$$

$$Violate_{Post} \equiv Pre_{R_P} \wedge (\bigwedge_{j \in [m]} \varphi_j) \wedge \neg Post_{R_P}$$

Theorem

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

Program	Verification condition	Structural Solving (MONA)			Data-constraint Solving (Z3 with QF-LIA)		
		in $STRAND_{dec}^{sem}$? (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
Sorted-list-reverse	before-loop	Yes	0.24	No	-	-	-
	in-loop	Yes	2.79	No	-	-	-
	after-loop	Yes	0.35	No	-	-	-
bubblesort	loop-if-if	Yes	7.70	No	-	-	-
	loop-if-else	Yes	6.83	No	-	-	-
	loop-else	Yes	2.73	Yes	8	No	0.02
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	8.84	No	-	-	-
	after-loop	Yes	1.76	No	-	-	-
left-rotate	bst-preserving	Yes	1.59	Yes	19	No	0.05

Conclusion

STRAND:

- A fundamental theory of combining decision procedures for graphs and data
- A semantic decidable fragment and a syntactic decidable fragment, both powerful enough for program verification
- Experimental results combining MONA and Z3 for analyzing heap programs.

Future Work:

- A fully-fledged engineering of an SMT solver for $STRAND_{dec}$
- An efficient non-automata theoretical decision procedure (unlike MONA) may yield more efficient decision procedures
- Decidable fragments of separation logic *with data*

Thank you!