Efficient Decision Procedures for Heaps using STRAND

P. Madhusudan and Xiaokang Qiu

University of Illinois at Urbana-Champaign, USA
{madhu,qiu2}@illinois.edu

Abstract. The STRAND [10] logic allows expressing structural properties of heaps combined with the data stored in the nodes of the heap. A semantic fragment of STRAND as well as a syntactically defined subfragment of it are known to be decidable [10]. The known decision procedure works by combining a decision procedure for MSO on trees (implemented by the tool MONA) and a decision procedure for the quantifier-free fragment of the data-theory (say, integers, and implemented using a solver like Z3).

The known algorithm for deciding the syntactically defined decidable fragment (which is the same as the one for the semantically defined decidable fragment) involves solving large MSO formulas over trees, whose solution is the main bottleneck in obtaining efficient algorithms. In this paper, we focus on the syntactically defined decidable fragment of STRAND, and obtain a new and more efficient algorithm. Using a set of experiments obtained from verification conditions of heap-manipulating programs, we show the practical benefits of the new algorithm.

1 Introduction

Several approaches to program analysis, like deductive verification, generating tests using constraint solving, abstraction, etc. have greatly benefited from the engineering of efficient SMT solvers, which currently provide automated decision procedures for a variety of quantifier-free theories, including integers, bit-vectors, arrays, uninterpreted functions, as well as *combinations* of these theories using the Nelson-Oppen method [15]. One of the most important kinds of reasoning in program verification that has evaded tractable decidable fragments is reasoning with dynamic heaps and the data contained in them.

Reasoning with heaps and data seems to call for decidable combinations of logics on *graphs* that model the heap structure (with heap nodes modeled as vertices, and field pointers as edges) with a logic on the data contained in them (like the quantifier-free theory of integers already supported by current SMT solvers). The primary challenge in building such decidable combinations stems from the *unbounded* number of nodes in the heap structures. This mandates the need for universal quantification in any reasonable logic in order to be able to refer to all the elements of the heap (e.g. to say a list is sorted, we need some form of universal quantification). However, the presence of quantifiers immediately

annuls the use of Nelson-Oppen combinations, and requires a new theory for combining unbounded graph theories with data.

Recently, we have introduced, along with Gennaro Parlato, a new logic called STRAND that combines heap structures and data [10]. We also identified a *semantically* defined decidable fragment of STRAND, called $\text{STRAND}^{sem}_{dec}$, as well as a syntactic decidable fragment, called STRAND_{dec}. The decision procedures for satisfiability for both the semantic and syntactic fragments were the same, and were based on (a) abstracting the data-predicates in the formula with Boolean variables to obtain a formula purely on graphs, (b) extracting the set of *minimal graphs* according to a satisfiability-preserving embedding relation that was completely agnostic to the data-logic, and is guaranteed to be minimal for the two fragments, and (c) checking whether any of the minimal models admits a data-extension that satisfies the formula, using a data-logic solver. We also showed that the decidable fragments can be used in the verification of pointer-manipulating programs. We implemented the decision procedure using MONA on tree-like data-structures for the graph logic and Z3 for quantifier-free arithmetic, and reported experimental results in Hoare-style deductive verification of certain programs manipulating data-structures.

In this paper, we concentrate on the *syntactic* decidable fragment of STRAND identified in the previous work [10], and develop new efficient decision procedures for it. The bottleneck in the decision procedure of the current methods for deciding STRAND is the phase that computes the set of all minimal models. This is done using monadic second-order logic (MSO) on trees, and is achieved by a *complex* MSO formula that has quantifier alternations and also includes adaptations of the STRAND formula *twice* within it. In experiments, this phase is clearly the bottleneck (for example, a verification condition for binary search trees takes about 30s while the time spent by Z3 on the minimal models is less than 0.5s).

We propose in this paper a new method to solve satisfiability for STRAND_{dec} using a notion called *small models*, which are not the precise set of minimal models but a slightly larger class of models. We show that the set of small models is always bounded, and also equisatisfiable (i.e. if there is any model that satisfies the STRAND_{dec} formula, then there is a data-extension of a small model that satisfies it). The salient feature of small models is that it can be expressed by a much simpler MSO formula that is *completely independent of the* STRAND_{dec} *formula*! The definition of small models depends only on the *signature* of the formula (in particular, the set of variables existentially quantified). Consequently, it does not mention any structural abstractions of the formula, and is much simpler to solve. This formulation of decidability is also a theoretical contribution, as it gives a much simpler alternative proof that the logic STRAND_{dec} is decidable.

We implement the new decision procedure, and show, using the same set of experiments as in [10], that the new procedure's performance is at least an order-of magnitude faster than the known one (in some examples, the new algorithm works even 1000 times faster).

In summary, this paper builds a new decision procedure for STRAND_{dec} that is based on new theoretical insights, and that is considerably faster than

the known decision procedure. We emphasize that the new decision procedure, though faster, is sound and complete in deciding the logic STRAND$_{dec}$. In developing STRAND, we have often been worried on the reliance of automata-theoretic decision procedures (like MONA), which tend to perform badly in practice for large examples. However, the decision procedure for STRAND crucially uses the minimal model property that seems to require solving MSO formulas, which in turn are currently handled most efficiently by automata-theoretic tools. This paper shows how the automata-theoretic decision procedures can be used on much more simplified formulas in building fast decision procedures for heaps using STRAND.

The paper is structured as follows. In Section 2, we give a high-level overview of STRAND followed by definitions that we need in this paper, including the notions of recursively defined data-structures, the notion of submodels, the definition of elastic relations, and the decidable fragment of STRAND. We do not describe the known decision procedure for the syntactic decidable fragment of STRAND— we refer the reader to [10]. Section 3 describes the new theoretical decision procedure for STRAND$_{dec}$ and compares it, theoretically, with the previously known decision procedure. Section 4 describes experimental comparisons of the new decision procedure with the old.

A brief note on notation: In the sequel, we will refer to the general STRAND logic [10] as STRAND*, and refer to the syntactic decidable fragment as STRAND (which is called STRAND$_{dec}$ in [10]).

2 Data-structures, submodels, elasticity and STRAND

In this section, we first give an informal overview of STRAND that will help understand the formal definitions. We then proceed to formally define the concepts of recursively defined data-structures, the notion of valid subsets of a tree, which in turn define submodels, define formulas that allow us to interpret an MSO formula on a submodel, and define the logic STRAND using elastic relations.

We refer the reader to [10] for more details, including the motivations for the design of the logic, how verification conditions can be formally extracted from code with pre- and post-conditions, as well as motivating examples.

2.1 An overview of STRAND

We give first an informal overview of the logic STRAND*, the syntactic decidable fragment STRAND, and how they can be used for verifying heap-manipulating programs, as set forth in [10].

We model heap structures as labeled directed graphs: the nodes of the graph correspond to heap locations, and an edge from n to n' labeled f represents the fact that the field pointer f of node n points to n'. The nodes in addition have *labels* associated to them; labels are used to signify special nodes (like NIL nodes) as well as to denote the program's pointer variables that point to them.

STRAND formulas are expressed over a particular class of heaps, called *recursively defined data-structures*. A (possibly infinite) set of recursively-defined data-structures is given by a tuple $(\psi_{Tr}, \psi_U, \{\alpha_a\}_{a\in L_v}, \{\beta_b\}_{b\in L_e})$. Here, ψ_{Tr} is an MSO formula on trees that defines a regular set of trees R, which forms the *backbone* skeletons of the structures, $\psi_U(x)$ is a monadic predicate expressed in MSO over trees that defines, for each tree, the subset of nodes that correspond to heap nodes, and the unary predicates α_a and binary predicates β_b, written in MSO over trees, identify the nodes labeled a and edges labeled b, respectively. The graphs that belong to the recursive data-structure are hence obtained by taking some tree T satisfying ψ_{Tr}, taking the subset of tree nodes of T that satisfy ψ_U as the nodes of the graph, and taking the a-labeled edges in the graph as those defined by E_a, for each $a \in \Sigma$.

The above way of defining graph-structures has many nice properties. First, it allows defining graphs in a way so that the MSO-theory on the graphs is decidable (by interpreting formulas on the underlying tree). Second, several naturally defined recursive data-structures in programs can be easily embedded in the above notation automatically. Intuitively, a recursive data-structure, such as a list of nodes pointing to trees or structs, has a natural skeleton which follows from the recursive data-type definition itself. In fact, *graph types* (introduced in [7]) are a simple textual description of recursive data-structures that are automatically translatable to our notation. Several structures including linked lists, doubly linked lists, cyclic and acyclic lists, trees, hierarchical combinations of these structures, etc., are definable using the above mechanism.

A STRAND* formula is defined over a set of recursively-defined data-structures, and is of the form $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$, where φ is a formula that combines MSO over graphs defined by a recursively-defined data structure, as well as logical constraints over the data stored in the heap nodes.

A *decidable* fragment of STRAND*, called STRAND (and called STRAND$_{dec}$ in [10]), is defined over a signature consisting of *elastic* relations and *non-elastic* relations, and allows formulas of the kind $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$ where φ has no further quantification, and where all non-elastic relations are restricted to the existentially quantified variables $\vec{x}$ (see below for formal definitions and see Fig. 2 for the syntax of the logic). Elastic relations have a technical definition: they are those that hold on any properly defined *sub-model* iff they hold in a model (see below for precise details). For example, on a tree, the *descendent* relation is an elastic relation, while the *child* relation is not.

In verifying programs, we require the user to give proof annotations that include pre-conditions and post-conditions and loop-invariants in STRAND$^*_{\exists,\forall}$, the fragment of STRAND* with a pure existential or universal prefix. The basic paths in the program hence become *assignments* and *assume* statements, and the invalidity of the Hoare-triple associated with the path can be reduced to the satisfiability of a *trail formula* in STRAND*. This formula, when it falls within the syntactic fragment STRAND, can be decided using the decision procedure set forth in this paper (see [10] for details on this construction).

It turns out that many annotations for heap-based programs actually fall in the syntactically defined fragment, and so do the verification conditions generated (in fact, all the conditions generated in the experiments in [10] fall into STRAND). The annotations fall into STRAND as several key properties, such as sortedness, the binary search-tree property, etc., can be stated in this restricted logic with a pure *universal* prefix. Furthermore, the verification condition itself often turns out to be expressible in STRAND, as the trail formula introduces variables for the footprint the basic path touches, but these variables are *existentially* quantified, and hence can be related using non-elastic relations (such as the next-node relation). Consequently, STRAND is a powerful logic to prove properties of heap manipulating programs.

We now formally define recursively defined data-structures, the notion of *valid subsets* of a tree (which allows us to define submodels), define elastic relations, and define the syntactic fragment STRAND.

2.2 Recursively defined data-structures

For any $k \in \mathbb{N}$, let $[k]$ denote the set $\{1, \ldots, k\}$. A k-ary tree is a set $V \subseteq [k]^*$, where V is non-empty and prefix-closed. We call $u.i$ the i'th child of u, for every $u, u.i \in V$, where $u \in [k]^*$ and $i \in [k]$. Let us fix a countable set of first-order variables FV (denoted by s, t, etc.), a countable set of set-variables SV (denoted by S, T, etc.), and a countable set of Boolean-variables BV (denoted by p, q, etc.). The syntax of the Monadic second-order (MSO) [20] formulas on k-ary trees is defined:

$$\delta ::= \; p \;\mid\; succ_i(s,t) \;\mid\; s = t \;\mid\; s \in S \;\mid\; \varphi \vee \varphi \;\mid\; \neg\varphi \;\mid\; \exists s.\varphi \;\mid\; \exists S.\varphi \;\mid\; \exists p.\varphi$$

where $i \in [k]$. The atomic formula $succ_i(s,t)$ holds iff t is the i'th child of s. Other logical symbols are interpreted in the traditional way.

Definition 1 (Recursively defined data-structures). *A class of* recursively defined data-structures *over a graph signature $\Sigma = (L_v, L_e)$ (where L_v and L_e are finite sets of labels) is specified by a tuple $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a \in L_v}, \{\beta_b\}_{b \in L_e})$ where ψ_{Tr} is an MSO sentence, ψ_U is a unary predicate defined in MSO, and each α_a and β_b are monadic and binary predicates defined using MSO, respectively, where all MSO formulas are over k-ary trees, for some $k \in \mathbb{N}$.* □

Let $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a \in L_v}, \{\beta_b\}_{b \in L_e})$ be a recursively defined data-structure and T be a k-ary Σ-labeled tree that satisfies ψ_{Tr}. Then $T = (V, \{E_i\}_{i \in [k]})$ defines a graph $Graph(T) = (N, E, \mu, \nu, L_v, L_e)$ as follows:

- $N = \{s \in V \mid \psi_U(s)$ *holds in* $T\}$
- $E = \{(s, s') \mid \psi_U(s)$ *and* $\psi_U(s')$ *hold, and* $\beta_b(s, s')$ *holds in* T *for some* $b \in L_e\}$
- $\mu(s) = \{a \in L_v \mid \psi_U(s)$ *holds and* $\alpha_a(s)$ *holds in* $T\}$
- $\nu(s, s') = \{b \in L_e \mid \psi_U(s)$ *and* $\psi_U(s')$ *hold and* $\beta_b(s, s')$ *holds in* $T\}$.

In the above, N denotes the nodes of the graph, E the set of edges, μ the labels on nodes, and ν the labels on edges. The class of graphs defined by $\mathcal{R}$ is the set $\mathit{Graph}(\mathcal{R}) = \{\mathit{Graph}(T) \mid T \models \psi_{Tr}\}$. These graphs are interpreted as heap structures.

We give an examples of modeling heap structures as recursively defined data-structures below.

Example 1. Binary trees are common data-structures. Two field pointers, l and r, point to the left and right children, respectively. If a node does not have a left (right) child, then the l (r) field points to the unique *NIL* node in the heap. Moreover, there is a node rt which is the root of the tree. Binary trees can be modeled as a recursively defined data-structure. For example, we can model the unique *NIL* node as the root of the tree, and model the actual nodes of the binary tree at the left subtree of the root (i.e. the tree under the left child of the root models rt). The right subtree of the root is empty. Binary trees can be modeled as $\mathcal{R}_{bt} = (\psi_{Tr}, \psi_U, \{\alpha_{rt}, \alpha_{nil}\}, \{\beta_l, \beta_r\})$ where

$$
\begin{aligned}
\psi_{Tr} &\equiv \exists y_1.\Big(\mathit{root}(y_1) \wedge \nexists y_2.\big(\mathit{succ}_r(y_1, y_2)\big)\Big)\\
\psi_U(x) &\equiv \mathit{true}\\
\alpha_{rt}(x) &\equiv \exists y.\big(\mathit{root}(y) \wedge \mathit{succ}_l(y, x)\big)\\
\alpha_{NIL}(x) &\equiv \mathit{root}(x)\\
\beta_l(x_1, x_2) &\equiv \exists y.\big(\mathit{root}(y) \wedge \mathit{leftsubtree}(y, x_1) \wedge \mathit{succ}_l(x_1, x_2)\big) \vee\\
&\quad \big(\mathit{root}(x_2) \wedge \nexists z.\mathit{succ}_l(x_1, z)\big)\\
\beta_r(x_1, x_2) &\equiv \exists y.\big(\mathit{root}(y) \wedge \mathit{leftsubtree}(y, x_1) \wedge \mathit{succ}_r(x_1, x_2)\big) \vee\\
&\quad \big(\mathit{root}(x_2) \wedge \nexists z.\mathit{succ}_r(x_1, z)\big)
\end{aligned}
$$

where the predicate $\mathit{root}(x)$ indicates whether x is the root of the backbone tree, and the relation $\mathit{leftsubtree}(y, x)$ ($\mathit{rightsubtree}(y, x)$) indicates whether x belongs to the subtree of the left (right) child of y. They can all be defined easily in MSO. As an example, Figure 1a shows a binary tree represented in $\mathcal{R}_{bt}$.

(a) T: a binary tree (b) S: a valid subset of T (shaded nodes)

Fig. 1: A binary tree example represented in $\mathcal{R}_{bt}$

2.3 Submodels

We first define *valid subsets* of a tree, with respect to a recursive data-structure.

Definition 2 (Valid subsets). *Let* $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a\in L_v}, \{\beta_b\}_{b\in L_e})$ *and* $T = (V, \lambda)$ *be a* Σ*-labeled tree that satisfies* ψ_{Tr}*, and let* $S \subseteq V$*. Then we say* S *is a* valid subset *of* V *if the following hold:*

- S *is non-empty, and least-common-ancestor closed (i.e. for any* $s, s' \in S$*, the least common ancestor of* s *and* s' *wrt* T *also belongs to* S*);*
- *The* subtree *defined by* S*, denoted by* $Subtree(T, S)$*, is the tree with nodes* S*, and where the* i*'th child of a node* $u \in S$ *is the (unique) node* $u' \in S$ *closest to* u *that is in the subtree rooted at the* i*'th child of* u*. (This is uniquely defined since* S *is least-common-ancestor closed.) We require that* $Subtree(T, S)$ *also satisfy* ψ_{Tr}*;*
- *for every* $s \in S$*, if* $\psi_U(s)$ *holds in* $Subtree(T, S)$*, then* $\psi_U(s)$ *holds in* T *as well;*
- *for every* $s \in S$*, for every* $a \in L_v$*,* $\alpha_a(s)$ *holds in* $Subtree(T, S)$ *iff* $\alpha_a(s)$ *holds in* T*.* □

Figure 1b shows a valid subset S of the binary tree representation T in Example 1. A tree $T' = (V', \lambda')$ is said to be a *submodel* of $T = (V, \lambda)$ if there is a valid subset S of V such that T' is isomorphic to $Subtree(T, S)$. Note that while unary predicates (α_a) are preserved in the submodel, the edge-relations (β_b) may be very different than its interpretation in the larger model.

Interpreting formulas on submodels. We define a transformation $tailor_X$ from an MSO formula on trees to another MSO formula on trees, such that for any MSO sentence δ on k-ary trees, for any tree $T = (V, \lambda)$ and any valid subset $X \subseteq V$, $Subtree(T, X)$ satisfies δ iff T satisfies $tailor_X(\delta)$. The transformation is given below, where we let $x \leq y$ mean that y is a descendent of x in the tree. The crucial transformations are the edge-formulas, which are interpreted as the edges of the subtree defined by X.

- $tailor_X(succ_i(s,t)) = \exists s'.\Big(E_i(s,s') \ \wedge \ s' \leq t \ \wedge \ \forall t'.\big((t' \in X \wedge s' \leq t') \Rightarrow t \leq t'\big)\Big)$, for every $i \in [k]$.
- $tailor_X(s = t) = (s = t)$
- $tailor_X(s \in W) = s \in W$
- $tailor_X(\delta_1 \vee \delta_2) = tailor(\delta_1) \vee tailor_X(\delta_2)$
- $tailor_X(\neg\delta) = \neg tailor_X(\delta)$
- $tailor_X(\exists s.\delta) = \exists s.\big(s \in X \wedge tailor_X(\delta)\big)$
- $tailor_X(\exists W.\delta) = \exists W.\big(W \subseteq X \wedge tailor_X(\delta)\big)$

Now by the definition of valid subsets, we define a predicate $ValidSubset(X)$ using MSO, where X is a free set variable, such that $ValidSubset(X)$ holds in a

tree $T = (V, \lambda)$ iff X is a valid subset of V (below, *lca(x,y,z)* stands for an MSO formula says that z is the least common ancestor of x and y in the tree).

$$\begin{aligned} \mathit{ValidSubset}(X) \;\equiv\; & \forall s,t,u.\Big(\big(s \in X \wedge t \in X \wedge \mathit{lca}(s,t,u)\big) \Rightarrow u \in X\Big) \\ & \wedge \mathit{tailor}_X(\psi_{Tr}) \wedge \Big(\forall s.\Big(s \in X \wedge \mathit{tailor}_X\big(\psi_U(s)\big)\Big) \Rightarrow \psi_U(s)\Big) \\ & \wedge \bigwedge_{a \in L_v} \Big[\; \forall s.\Big(s \in X \Rightarrow \Big(\mathit{tailor}_X\big(\alpha_a(s)\big) \Leftrightarrow \alpha_a(s)\Big)\Big) \;\Big] \end{aligned}$$

2.4 Elasticity and STRAND

Elastic relations are relations of the recursive data-structure that satisfy the property that a pair of nodes satisfy the relation in a tree iff they also satisfy the relation in any valid subtree. Formally,

Definition 3 (Elastic relations). *Let $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a \in L_v}, \{\beta_b\}_{b \in L_e})$, and let $b \in L_e$ be an edge label. Then the relation E_b (defined by β_b) is* elastic *if for every tree $T = \{V, \lambda\}$ satisfying ψ_{Tr}, for every valid subset S of V, and for every pair of nodes u, v in the model $M' = \mathit{Graph}(\mathit{Subtree}(T, S))$, $E_b(u, v)$ holds in M' iff $E_b(u, v)$ holds in $\mathit{Graph}(T)$.* □

For example, let $\mathcal{R}$ be the class of binary trees, the *left-descendent* relation relating a node with any of the nodes in the tree subtended from the left child, is elastic, because for any binary tree T and any valid subset of S containing nodes x and y, if y is in the left branch of x in T, y is also in the left branch of x in the subtree defined by S, and vise versa. However, the *left-child* relation is non-elastic. Consider a binary tree T in which y is in the left branch of x but not the left child of x, then $S = \{x, y\}$ is a valid subset, and y is the left child of x in $\mathit{Subtree}(T, S)$.

It turns out that elasticity of E_b can also be expressed by the following MSO formula

$$\begin{aligned} \psi_{Tr} \;\Rightarrow\; \forall X\; \forall u\; \forall v.\; \Big[\; \Big(& \mathit{ValidSubset}(X) \wedge u{\in}X \wedge v{\in}X \wedge \\ & \mathit{tailor}_X\big(\psi_U(u)\big) \wedge \mathit{tailor}_X\big(\psi_U(v)\big)\Big) \\ \Rightarrow\; & \Big(\beta_b(u,v) \Leftrightarrow \mathit{tailor}_X\big(\beta_b(u,v)\big)\Big) \;\Big] \end{aligned}$$

E_b is elastic iff the above formula is valid over all trees. Hence, we can *decide* whether a relation is elastic or not, by checking the validity of the above formula over k-ary Σ-labeled trees.

For the rest of this paper, let us fix a class of recursively defined data-structures $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a \in L_v}, \{\beta_b\}_{b \in L_e})$, with $L_e^E \subseteq L_e$ the set of elastic edge relations, and let $L_e^{NE} = L_e \backslash L_e^E$ the non-elastic edge relations. All notations used are with respect to $\mathcal{R}$.

Formula	$\psi ::= \exists x.\psi \mid \omega$
$\forall$Formula	$\omega ::= \forall y.\omega \mid \varphi$
QFFormula	$\varphi ::= \gamma(e_1, \ldots, e_n) \mid Q_a(v) \mid E_b(v, v') \mid E_{b'}(x, x')$ $\mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$
Expression	$e ::= data(x) \mid data(y) \mid c \mid g(e_1, \ldots, e_n)$
$\exists$DVar	$x \in Loc$
$\forall$DVar	$y \in Loc$
DVar	$v ::= x \mid y$
Constant	$c \in Sig(\mathcal{D})$
Function	$g \in Sig(\mathcal{D})$
$\mathcal{D}-$Relation	$\gamma \in Sig(\mathcal{D})$
E $-$ Relation	$b \in L_e^E$
NE $-$ Relation	$b' \in L_e^{NE}$
Predicate	$a \in L_v$

Fig. 2: Syntax of STRAND

STRAND* (called STRAND in [10]) is a two-sorted logic interpreted on program heaps with both locations and the data stored in them. STRAND formulas are of the form $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$, where $\vec{x}$ and $\vec{y}$ are $\exists$DVar and $\forall$DVar, respectively, (we also refer to both as DVar), φ is a formula that combines structural constraints as well as data-constraints, but their data-constraints are only allowed to refer to $\vec{x}$ and $\vec{y}$. STRAND* is an expressive logic, allowing complex combinations of structural and data constraints. This paper focuses on a decidable fragment STRAND. Given a recursively defined data-structure $\mathcal{R}$ and a first-order theory $\mathcal{D}$, the syntax of STRAND is presented in Figure 2.

Intuitively, STRAND formulas are of the kind $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$, where φ is quantifier-free and combines both data-constraints and structural constraints, with the important restriction that *the atomic relations involving universally quantified variables be only elastic relations.*

3 The new decision procedure for STRAND

The decision procedure for STRAND presented in [10] worked as follows. Given a STRAND formula ψ over a class of recursively defined data-structures $\mathcal{R}$, we first construct a pure MSO formula on k-ary trees $MinModel_\psi$ that captures the subset of trees that are minimal with respect to an equi-satisfiability preserving embedding relation. This assures that if the formula ψ is satisfiable, then it is satisfiable by a data-extension of a minimal model (a minimal model is a model satisfying $MinModel_\psi$). Furthermore, this set of minimal models was guaranteed to be finite. The decision procedure is then to do a simple analysis on the tree automaton accepting all minimal models, to determine the maximum height h

of all minimal trees, and then query the data-solver as to whether any tree of height bounded by h satisfies the STRAND formula.

In this paper, we follow a similar approach, but we replace the notion of minimal models with a new notion called *small models*. Given a STRAND formula $\psi = \exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$ over a class of recursively defined data-structures $\mathcal{R} = (\psi_{Tr}, \psi_U, \{\alpha_a\}_{a\in L_v}, \{\beta_b\}_{b\in L_e})$, the MSO formula $\mathit{SmallModel}(\vec{x})$ is defined on k-ary trees, with free variables $\vec{x}$. Intuitively, $\mathit{SmallModel}(\vec{x})$ says that there does not exist a nontrivial valid subset X such that X contains $\vec{x}$, and further satisfies the following: for every non-elastic relation possibly appearing in $\varphi(\vec{x},\vec{y})$, it holds in $\mathit{Graph}(T)$ iff it holds in $\mathit{Graph}(\mathit{Subtree}(T,X))$. Since the evaluations of other atomic formulas, including elastic relations and data-logic relations, are all preserved, we can prove that $\mathit{SmallModel}(\vec{x})$ is equisatisfiable to the structural constraints in ψ, but has only a finite number of models. The formula $\mathit{SmallModel}(\vec{x})$ is defined as follows:

$$\begin{aligned}
\mathit{SmallModel}(\vec{x}) \quad \equiv \quad & \psi_{Tr} \ \wedge \bigwedge_{x\in\vec{x}} \psi_U(x) && (\zeta)\\
\wedge \quad & \neg\exists X.\Big(\mathit{ValidSubset}(X) \ \wedge \bigwedge_{x\in\vec{x}} (x\in X) \ \wedge \\
& \qquad \exists s.(s\in X) \ \wedge \ \exists s.(s\notin X) \ \wedge && (\eta)\\
& \qquad \bigwedge_{b\in L_e^{NE}, x,x'\in\vec{x}} \Big(\beta_b(x,x') \Leftrightarrow \mathit{tailor}_X\big(\beta_b(x,x')\big)\Big)\Big)
\end{aligned}$$

Note that the above formula *does not depend* on the STRAND formula ψ at all, except for the set of existentially quantified variables $\vec{x}$.

Our proof strategy is now as follows. We show two technical results:

(a) For any $\vec{x}$, $\mathit{SmallModel}(\vec{x})$ has only finitely many models (Theorem 1 below). This result is independent of the fact that we are dealing with STRAND formulas.

(b) A STRAND formula ψ with existentially quantified variables $\vec{x}$ has a model iff there is some data-extension of a model satisfying $\mathit{SmallModel}(\vec{x})$ that satisfies ψ (Theorem 2 below).

The above two establish the correctness of the decision procedure. Given a STRAND formula ψ, with existential quantification over $\vec{x}$, we can compute a tree-automaton accepting the set of all small models (i.e. the models of $\mathit{SmallModel}(\vec{x})$), compute the maximum height h of the small models, and then query the data-solver as to whether there is a model of height at most h *with data* that satisfies ψ.

We now prove the two technical results.

Theorem 1. *For any recursively defined data-structure $\mathcal{R}$ and any finite set of variables $\vec{x}$, the number of models of $\mathit{SmallModel}(\vec{x})$ is finite.*

Proof. Fix a recursively defined data-structure $\mathcal{R}$ and a finite set of variables $\vec{x}$. It is sufficient to show that for any model T of *SmallModel*($\vec{x}$), the size of T is bounded.

We first split *SmallModel*($\vec{x}$) into two parts: let ζ be the first two conjuncts, i.e., $\psi_{Tr} \wedge \bigwedge_{x \in \vec{x}} \psi_U(x)$, and η be the last conjunct.

Recall the classic logic-automata connection: for any MSO formula $\theta(\vec{y}, \vec{Y})$ with free first-order variables $\vec{y}$ and free set-variables $\vec{Y}$, we can construct a tree-automaton that precisely accepts those trees with encodings of the valuation of $\vec{y}$ and $\vec{Y}$ as extra labels that satisfy the formula θ [20].

Construct a deterministic (bottom-up) tree automaton A_ζ that accepts precisely the models satisfying $\zeta(\vec{x})$, using this classic logic-automata connection [20]. Also, for each *non-elastic* edge label $b \in L_r^{NE}$, and each pair of variables $x, x' \in \vec{x}$, let $A_{b,x,x'}$ be a deterministic (bottom-up) tree automaton that accepts the models of the formula $\beta_b(x, x')$.

(a) T with the valid subset X (shaded dark) (b) *Subtree*(T, X)

Fig. 3: A valid subset X that falsifies β

It is clear that T is accepted by A_ζ, while $A_{b,x,x'}$, for each b, x, x', either accepts or rejects T. Construct the *product* of the automaton A_ζ and all automata $A_{b,x,x'}$, for each b, x, x', with the acceptance condition derived solely from A_ζ; call this automaton B; then B accepts T.

If the accepted run of B on T is r, then we claim that r is *loop-free* (a run of a tree automaton is loop-free if for any path of the tree, there are no two nodes in the path labeled by the same state). Assume not. Then there must be two different nodes p_1, p_2 such that p_2 is in the subtree of p_1, and both p_1 and p_2 are labeled by the same state q in r. Then we can *pump down* T by merging p_1 and p_2. The resulting tree is accepted by A_T as well. Consider the subset X of T that consists of those remaining nodes, as shown in Figure 3. It is not hard to see X is a nontrivial valid subset of T. Also for each $b \in L_r^{NE}$ and each $x, x' \in \vec{x}$, since the run of $A_{b,x,x'}$ ends up in the same state on reading the subtree corresponding to X, $\beta_b(x, x')$ holds in T iff $\beta_b(x, x')$ holds in *Subtree*(T, X). Thus

X is a valid subset of T that acts to falsify η, which contradicts our assumption that T satisfies $\zeta \wedge \eta$.

Since r is loop-free, the height of T is bounded by the number of states in B. □

We now show that the small models define an adequate set of models to check for satisfiability.

Theorem 2. *Let $\mathcal{R}$ be a recursively defined data-structure and let $\psi = \exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$ be a* STRAND *formula. If ψ is satisfiable, then there is a model $\mathcal{M}$ of ψ and a model T of SmallModel($\vec{x}$) such that Graph(T) is isomorphic to the graph structure of $\mathcal{M}$.*

Proof. Let ψ be satisfiable and let $\mathcal{M}$ satisfy ψ. Then there is an assignment I of $\vec{x}$ over the nodes of $\mathcal{M}$ under which $\forall\vec{y}\varphi(\vec{x},\vec{y})$ is satisfied.

Let T be the backbone tree of the graph model of $\mathcal{M}$, and further let us add an extra label over T to denote the assignment I to $\vec{x}$.

Let us, without loss of generality, assume that T is a *minimal* tree; i.e. T has the least number of nodes among all models satisfying ψ.

We claim that T satisfies *SmallModel*($\vec{x}$) under the interpretation I.

Assume not, i.e., T does not satisfy *SmallModel*($\vec{x}$) under I. Since T under I satisfies ζ, it must not satisfy η. Hence there exists a strict valid subset of nodes, X, such that every non-elastic relation holds over every pair of variables in $\vec{x}$ the same way in T as it does on the subtree defined by X.

Let $\mathcal{M}'$ be the model obtained by taking the graph of the subtree defined by X as the underlying graph, with data at each obtained node inherited from the corresponding node in $\mathcal{M}$. We claim that $\mathcal{M}'$ satisfies ψ as well, and since the tree corresponding to $\mathcal{M}'$ is a strict subtree of T, this contradicts our assumption on T.

We now show that the graph of the subtree defined by X has a data-extension that satisfies ψ.

In order to satisfy ψ, we take the interpretation of each $x \in \vec{x}$ to be the node in $\mathcal{M}'$ corresponding to $I(x)$. Now consider any valuation of $\vec{y}$. We will show that every *atomic* relation in φ holds in $\mathcal{M}$ in precisely *the same way* as it does on $\mathcal{M}'$; this will show that φ holds in $\mathcal{M}$ iff φ holds in $\mathcal{M}'$, and hence that φ holds in $\mathcal{M}'$.

By definition, an atomic relation τ could be a unary predicate, an elastic binary relation, a non-elastic binary relation, or an atomic data-relation. If τ is a unary predicate $Q_a(v)$ (where $v \in \vec{x} \cup \vec{y}$), then by definition of submodels (and valid subsets), τ holds in $\mathcal{M}'$ iff τ holds in $\mathcal{M}$. If τ is an elastic relation $E_b(v_1, v_2)$, by definition of elasticity, τ holds in $\mathcal{M}$ iff $\beta_b(v_1, v_2)$ holds in T iff $\beta_b(v_1, v_2)$ holds in *Subtree*(T, X) iff $\tau(v_1, v_2)$ holds in $\mathcal{M}'$. If τ is a non-elastic relation, it must be of form $E_b(x, x')$ where $x, x' \in \vec{x}$. By the properties of X established above, it follows that $E_b(x, x')$ holds in $\mathcal{M}'$ iff $E_b(x, x')$ holds in $\mathcal{M}$. Finally, if τ is an atomic data-relation, since the data-extension of $\mathcal{M}'$ is inherited from $\mathcal{M}$, the data-relation holds in $\mathcal{M}'$ iff it holds in $\mathcal{M}$.

The contradiction shows that $\mathcal{M}$ is a small model. □

The above results can be used to even *pre-compute* the bounds on sizes of the structural models for a fixed recursive data-structure and for various numbers of existentially quantified variables, and *completely* avoid the structural phase altogether. We can even establish these bounds *analytically* (and manually), without the use of a solver, for some data-structures. For instance, over trees, with non-elastic relations *left-child* and *right-child*, and other elastic relations, it is not hard to see that a STRAND formula is satisfiable iff it is satisfiable by a tree of size at most $2n$, where n is the number of existentially quantified variables in the formula.

3.1 Comparison with earlier known decision procedure

We now compare the new decision procedure, technically, with the earlier known decision procedure for STRAND [10], which was also the decision procedure for the semantic decidable fragment $\text{STRAND}^{sem}_{dec}$.

The known decision procedure for STRAND [10] worked as follows. Given a STRAND formula, we first eliminate the leading existential quantification, by absorbing it into the signature, using new constants. Then, for the formula $\forall\vec{y}\varphi$, we define a *structural abstraction* of φ, named $\widehat{\varphi}$, where all data-predicates are replaced uniformly by a set of Boolean variables $\vec{p}$. A model that satisfies $\widehat{\varphi}$, for every valuation of $\vec{y}$, using some valuation of $\vec{p}$ is said to be a *minimal model* if it has no proper submodel that satisfies $\widehat{\varphi}$ under the *same* valuation $\vec{p}$, for every valuation of $\vec{y}$. Intuitively, this ensures that if the model can be populated with data in some way so that $\forall\vec{y}\varphi$ is satisfied, then the submodel satisfy the formula $\forall\vec{y}\varphi$ as well, by inheriting the same data-values from the model.

It turns out that for any STRAND formula, the number of minimal models (with respect to the submodel relation) is *finite* [10]. Moreover, we can capture the class of all minimal models using an MSO formula of the following form:

$$\begin{aligned} \mathit{MinModel} = \neg\exists X.\Big[\ & \mathit{ValidSubset}(X)\ \wedge\ \exists s.(s \in X) \wedge \exists s.(s \notin X)\ \wedge \\ & \forall\vec{y}\ \forall\vec{p}\ \Big(\Big(\wedge_{y\in\vec{y}}\ \big(y \in X \wedge \psi_U(y)\big) \wedge \mathit{interpret}(\widehat{\varphi}(\vec{y},\vec{p}))\Big) \\ & \qquad \Rightarrow \mathit{tailor}_X\big(\mathit{interpret}(\widehat{\varphi}(\vec{y},\vec{p}))\big)\Big)\ \Big] \end{aligned}$$

The above formula intuitively says that a model is minimal if there is no valid non-trivial submodel such that for all possible valuations of $\vec{y}$ in the submodel, and all possible valuations of $\vec{p}$, the model satisfies the structural abstraction $\widehat{\varphi}(\vec{y},\vec{p})$, then so does the submodel.

We can enumerate all the minimal models (all models satisfying the above formula), and using a data-constraint solver, ask whether any of them can be extended with data to satisfy the STRAND formula. Most importantly, if none of them can, we know that the formula is unsatisfiable (for if there was a model that satisfies the formula, then one of the minimal models will be a submodel that can inherit the data values and satisfy the formula).

Comparing the above formula to the formula for *SmallModel*, notice that the latter is incredibly simpler as it does not refer to the STRAND formula (i.e. φ) at all! The *SmallModel* formula just depends on the *set of existentially quantified variables* and the non-elastic relations in the signature. In contrast, the formula above for *MinModel* uses adaptations of the STRAND formula φ *twice* within it. In practice, this results in a very complex formula, as the verification conditions get long and involved, and this results in poor performance by the MSO solver. In contrast, as we show in the next section, the new procedure results in simpler formulas that get evaluated significantly faster in practice.

4 Experiments

Program	Verification condition	Minimal Model computation (old Alg. [10])		Small Model computation (new Alg.)		Data-constraint solving (Z3, QF-LIA)
		Max. BDD size	Time(s)	Max. BDD size	Time(s)	Old/New Time (s)
sorted-list-search	before-loop	10009	0.34	**540**	**0.01**	-
	in-loop	17803	0.59	**12291**	**0.14**	-
	after-loop	3787	0.18	**540**	**0.01**	-
sorted-list-insert	before-head	59020	1.66	**242**	**0.01**	0.02/0.02
	before-loop	15286	0.38	**595**	**0.01**	-
	in-loop	135904	4.46	**3003**	**0.03**	-
	after-loop	475972	13.93	**1250**	**0.01**	0.02/0.03
sorted-list-insert-error	before-loop	14464	0.34	**595**	**0.01**	0.02/0.02
sorted-list-reverse	before-loop	2717	0.24	**1155**	**0.01**	-
	in-loop	89342	2.79	**12291**	**0.14**	-
	after-loop	3135	0.35	**1155**	**0.01**	-
bubblesort	loop-if-if	179488	7.70	**73771**	**1.31**	-
	loop-if-else	155480	6.83	**34317**	**0.48**	-
	loop-else	95181	2.73	**7017**	**0.07**	0.02/0.04
bst-search	before-loop	9023	5.03	**1262**	**0.31**	-
	in-loop	26163	32.80	**3594**	**2.43**	0.02/0.11
	after-loop	6066	3.27	**1262**	**0.34**	-
bst-insert	before-loop	3485	1.34	**1262**	**0.34**	-
	in-loop	17234	9.84	**1908**	**1.38**	-
	after-loop	2336	1.76	**1807**	**0.46**	-
left-/right-rotate	bst-preserving	**1086**	1.59	1510	**0.48**	0.05/0.14
Total			98.15		**7.99**	0.15/0.36

Fig. 4: Results of program verification.
Experiments available at `http://www.cs.uiuc.edu/~qiu2/strand/`

In this section, we demonstrate the efficiency of the new decision procedure for STRAND by checking verification conditions of several heap-manipulating

programs, and comparing them to the decision procedure in [10]. These examples include list-manipulating and tree-manipulating programs, including searching a sorted list, inserting into a sorted list, in-place reversal of a sorted list, the bubble-sort algorithm, searching for a key in a binary search tree, inserting into a binary search tree, and doing a left- or right-rotate on a binary search tree.

For all these examples, a set of partial correctness properties including both structural and data requirements is checked. For example, assuming a node with value k exists, we check if both `sorted-list-search` and `bst-search` return a node with value k. For `sorted-list-insert`, we assume that the inserted value does not exist, and check if the resulting list contains the inserted node, and the sortedness property continues to hold. In the program `bst-insert`, assuming the tree does not contain the inserted node in the beginning, we check whether the final tree contains the inserted node, and the binary-search-tree property continues to hold. In `sorted-list-reverse`, we check if the output list is a valid list that is reverse-sorted. The code for `bubblesort` is checked to see if it results in a sorted list. And the `left-rotate` and `right-rotate` codes are checked to see whether they maintain the binary search-tree property.

In the structural solving phase using MONA, when a STRAND formula ψ is given, we further optimize the formula $\mathit{SmallModel}(\vec{x})$ with respect to ψ for better performance, as follows. First, a sub-formula $\beta_b(x, x') \Leftrightarrow \mathit{tailor}_X(\beta_b(x, x'))$ appears in the formula only if the atomic formula $E_b(x, x')$ appears in ψ. Moreover, if $E_b(x, x')$ only appears positively, we use $\beta_b(x, x') \Rightarrow \mathit{tailor}_X(\beta_b(x, x'))$ instead; similarly if $E_b(x, x')$ occurs only negatively, then we use $\beta_b(x, x') \Leftarrow \mathit{tailor}_X(\beta_b(x, x'))$ instead. This is clearly sound.

Figure 4 shows the comparison of the two decision procedures on checking the verification conditions. The results for both procedures were conducted on the same 2.2GHz, 4GB machine running Windows 7. We also report the size of the largest intermediate BDD and the time spent by MONA.

The experimental results show a magnitude of speed-up, with some examples (like `sorted-list-insert/after-loop`) giving even a 1000X speedup. The peak BDD sizes are also considerably smaller, in general, using the new algorithm.

Turning to the *bounds* computed on the minimal/small models, the two procedures generate the same bounds (i.e. the same lengths for lists and the same heights for trees) on all examples, except `bst-search-in-loop` and `left-rotate`. The condition `bst-search-in-loop` gave a maximal height of 5 with the old procedure and maximal height 6 in the new, while `left-rotate` generates maximal height 5 in the old procedure and 6 in the new. However, the larger bounds did not affect the data solving phase by any significant degree (at most by $0.1s$)

The experiments show that the new algorithm is considerably more efficient than the earlier known algorithm on this set of examples.

5 Related Work

Apart from [10], the work closest to ours is PALE [13], which is a logic on heaps structures but not data, and uses MSO and MONA [6] to decide properties of

heaps. TASC [5] is similar but generalizes to reason balancedness in the cases of AVL and red-black trees. First-order logics with axiomatizations of the reachability relation (which cannot be expressed in FOL) have been proposed: axioms capturing *local* properties [12], a logic on regular patterns that is decidable [21], among others.

The logic in HAVOC, called LISBQ [9], offers reasoning with generic heaps combined with an arbitrary data-logic. The logic has restricted reachability predicates and universal quantification, but is syntactically severely curtailed to obtain decidability. Though the logic is not very expressive, it is extremely efficient, as it uses no structural solver, but translates the structure-solving also to the (Boolean aspect) of the SMT solver. The logic CSL [4] is defined in a similar vein as HAVOC, with similar sort-restrictions on the syntax, but generalizes to handle doubly-linked lists, and allows size constraints on structures. As far as we know, neither HAVOC nor CSL can express the verification conditions of searching a binary search tree. The work reported in [3] gives a logic that extends an LTL-like syntax to define certain decidable logic fragments on heaps.

The inference rule system proposed in [16] for reasoning with restricted reachability does not support universal quantification and cannot express disjointness constraints, but has an SMT solver based implementation [17]. Restricted forms of reachability were first axiomatized in early work by Nelson [14]. Several mechanisms without quantification exist, including the work reported in [18, 1]. Kuncak's thesis describes automatic decision procedures that approximate higher-order logic using first-order logic, through approximate logics over sets and their cardinalities [8].

Finally, separation logic [19] is a convenient logic to express heap properties of programs, and a decidable fragment (without data) on lists is known [2]. However, not many extensions of separation logics support data constraints (see [11] for one that does).

6 Conclusions

The decision procedures for STRAND use a structural phase that computes a set of minimal structural models in a completely data-logic agnostic manner [10]. The new decision procedure set forth in this paper gives a way of computing an equisatisfiable finite set of structural models that is even agnostic to the STRAND formula. This yields a much simpler decision procedure, in theory, and a much faster decision procedure, in practice.

We are emboldened by the very minimal reliance on the structural solver (MONA) and we believe that the approach described in this paper is ready to be used for generating unit test inputs for heap-manipulating methods using symbolic constraint solving. Implementing such a procedure, as well as implementing a fully-fledged solver for STRAND, are interesting future directions to pursue.

Acknowledgements. This work is partially funded by NSF CAREER award #0747041.

References

1. Balaban, I., Pnueli, A., Zuck, L.D.: Shape analysis of single-parent heaps. In: VMCAI'07. LNCS, vol. 4349, pp. 91–105. Springer (2007)
2. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: FSTTCS'04. LNCS, vol. 3328, pp. 97–109. Springer (2004)
3. Bjørner, N., Hendrix, J.: Linear functional fixed-points. In: CAV'09. LNCS, vol. 5643, pp. 124–139. Springer (2009)
4. Bouajjani, A., Dragoi, C., Enea, C., Sighireanu, M.: A logic-based framework for reasoning about composite data structures. In: CONCUR'09. LNCS, vol. 5710, pp. 178–195. Springer (2009)
5. Habermehl, P., Iosif, R., Vojnar, T.: Automata-based verification of programs with tree updates. Acta Informatica 47(1), 1–31 (2010)
6. Klarlund, N., Møller, A.: MONA. BRICS, Department of Computer Science, Aarhus University (January 2001), available from `http://www.brics.dk/mona/`.
7. Klarlund, N., Schwartzbach, M.I.: Graph types. In: POPL'93. pp. 196–205. ACM (1993)
8. Kuncak, V.: Modular Data Structure Verification. Ph.D. thesis, Massachusetts Institute of Technology (2007)
9. Lahiri, S., Qadeer, S.: Back to the future: revisiting precise program verification using SMT solvers. In: POPL'08. pp. 171–182. ACM (2008)
10. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: POPL'11. pp. 611–622. ACM (2011)
11. Magill, S., Tsai, M.H., Lee, P., Tsay, Y.K.: THOR: A tool for reasoning about shape and arithmetic. In: CAV'08. LNCS, vol. 5123, pp. 428–432. Springer (2008)
12. McPeak, S., Necula, G.C.: Data structure specifications via local equality axioms. In: CAV'05. LNCS, vol. 3576, pp. 476–490. Springer (2005)
13. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: PLDI'01. pp. 221–231. ACM (2001)
14. Nelson, G.: Verifying reachability invariants of linked structures. In: POPL'83. pp. 38–47. ACM (1983)
15. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. 1, 245–257 (1979)
16. Rakamaric, Z., Bingham, J.D., Hu, A.J.: An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In: VMCAI'07. LNCS, vol. 4349, pp. 106–121. Springer (2007)
17. Rakamarić, Z., Bruttomesso, R., Hu, A.J., Cimatti, A.: Verifying heap-manipulating programs in an SMT framework. In: ATVA'07. LNCS, vol. 4762, pp. 237–252. Springer (2007)
18. Ranise, S., Zarba, C.: A theory of singly-linked lists and its extensible decision procedure. In: SEFM'06. pp. 206–215. IEEE-CS (2006)
19. Reynolds, J.: Separation logic: a logic for shared mutable data structures. In: LICS'02. pp. 55–74. IEEE-CS (2002)
20. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, pp. 389–456. Springer (1997)
21. Yorsh, G., Rabinovich, A.M., Sagiv, M., Meyer, A., Bouajjani, A.: A logic of reachable patterns in linked data-structures. In: FoSSaCS'06. LNCS, vol. 3921, pp. 94–110. Springer (2006)