# Verification of Multi-Linked Heaps 

Ittai Balaban ${ }^{\text {a }}$, Amir Pnueli, Yaniv Sa’ar ${ }^{\text {b }}$, Lenore D. Zuck ${ }^{\text {c }}$<br>${ }^{a}$ New York University, New York<br>${ }^{b}$ Weizmann Institute of Science, Israel<br>${ }^{c}$ University of Illinois at Chicago


#### Abstract

We define the class of single-parent heap systems, which rely on a singly-linked heap in order to model destructive updates on tree structures. This encoding has the advantage of relying on a relatively simple theory of linked lists in order to support abstraction computation. To facilitate the application of this encoding, we provide a program transformation that, given a program operating on a multi-linked heap without sharing, transforms it into one over a single-parent heap. It is then possible to apply shape analysis by predicate and ranking abstraction. The technique has been successfully applied on examples with lists (reversal and bubble sort) and trees with of fixed arity (balancing of, and insertion into, a binary sort tree).


Keywords: Heaps, Shape Analysis, Verification, Abstraction, Ranking Abstraction, Small Model, Model Checking, Termination, Trees, Lists.

## 1. Introduction

This paper is based on research reported in [3, 5].
The goal of shape analysis is to analyze properties of programs that perform destructive updating on dynamically allocated storage (heaps) [21]. Programs manipulating heap structures can be viewed as parameterized in the number of heap nodes, or, alternatively, the memory size.

This paper presents an approach for shape analysis based on predicate abstraction that allows for analysis of functional properties such as safety and liveness. The abstraction used does not require any abstract representation of the heap nodes (e.g. abstract shapes), but rather, requires only reachability relations between the program variables.

States are abstracted using a predicate base that contains reachability relations among program variables pointing into the heap. The computation of the abstract states and transition relation is precise, automatic, and does not require the use of a theorem prover. Rather, we use a small model theorem to identify a truncated (small) finite-state version of the program whose abstraction is identical to the abstraction of the unbounded-heap version of the same program. The abstraction of the finite-state version is then computed by BDD techniques.

For proving liveness properties, we augment the original system by a well-founded ranking function, which is then abstracted together with the system. Well-foundedness
is abstracted into strong fairness (compassion). We demonstrate the power of the ranking abstraction method and its advantages over direct use of ranking functions in a deductive verification of the same property, independent of its application to shapeanalysis examples. We show how various predicate abstractions can be used to establish various safety properties, and how, for each program, one of the abstractions can be augmented with a progress monitor to establish termination.

We first introduce single-parent heaps, which are finite heap systems specialized for representing trees and lists. We propose a framework for shape analysis of singleparent heaps based on a small model property of a restricted class of first order assertions with transitive closure. Extending this framework to allow for heaps with multiple links per node entails extending the assertional language and proving a stronger small model property. At this point, it is not clear whether such a language extension is decidable (see [17, 19] for relevant results).

This paper deals with verification of programs on multi-linked heaps that consist only of trees of bounded arity, which perform destructive updates of heaps. We bypass the need to handle trees directly by transforming heaps consisting of multiple trees into single-parent heaps that are structures consisting of singly-linked lists (possibly with shared suffixes). This is accomplished by "reversing" the parent-to-child edges of the trees populating the heap, as well as associating scalar data with nodes. Fig. 1(a) and Fig. 1(b) together demonstrate the transformation of a multi-linked heap that consists of a binary tree to its single-parent counterpart. In the latter graph, edges are directed from children to parents, and each child is annotated with boolean information denoting whether it is a left or right child.

(b) The Corresponding Single-Parent Heap

Figure 1: Multi-Linked to Single-Parent Heap Transformation
Verification of temporal properties of multi-linked heap systems can be performed as follows: Given a multi-linked system and a temporal property, the system and property are (automatically) transformed into their single-parent counterparts. Then, a counter-example-guided predicate- (and possibly ranking-) abstraction refinement method ([4]) is applied. If a counter-example (on the transformed system) is produced, it is automatically mapped into a counter-example of the original (multi-linked) system.

The rest of this paper is organized as follows: After we discuss related work, we present in Section 2 two motivating examples which are used as running examples throughout the paper. In Section 3 we present the formal framework, the single-parent heap model, and the assertianal language. In Section 4 we introduce multi-linked heaps
and formally show how to translate programs, as well as assertions, over them into the single-parent model. In Section 5 we overview the idea of abstraction and present a small model property that allows to predicate-abstract single-parent heap programs based on their instantiations to small heaps. In Section 6 we overview the method of ranking abstraction for proving termination, present several ranking functions that we show are useful for proving termination of programs that manipulate heaps. In Section 7 we describe examples demonstrating the power of the method, and we conclude in Section 8.

All our examples have been automatically tested using JTLV ([26]). The code is available in http://shape-analysis.ysaar.net/

## Related Work

The shape analysis method presented in this paper combines the frameworks of predicate abstraction [29], model checking [14, 16], and ranking abstraction [4], with a decision procedure over a logic of "tree-like" data structures (with limited cycles), and their mutation. The decision procedure itself is based on two elements: A program transformation, and a small model theorem for a logic of mutation of single-parent heaps.

Numerous frameworks exist for analyzing singly-linked heaps, e.g., [24, 28, 9, 12, 10]. In contrast to our framework, all of these assume that programs access only those heap cells that are reachable from program variables. While this is a reasonable assumption for most programs, it does not hold for the programs generated by our transformation, which manipulate heaps in which the links between heap cells are reversed. Therefore, these frameworks cannot be used to analyze single-parent heapmanipulating programs.

The correspondence between tree structures and singly-linked structures is the basis of the proof of decidability of first-order logic with one function symbol in [11]. More generally, the observation that complex data structures with regular properties can be reduced to simpler structures has been utilized in [23, 20, 25, 31]. However, it is not always straightforward to apply, and, to our knowledge, has not been applied in the context of predicate abstraction. Several assumptions that hold true in analysis of "conventional" programs over singly-linked heaps (e.g., C- or Pascal-programs), cannot be relied upon when reducing trees to lists. For example, the number of roots of the heap is no longer bounded by the number of program variables.

The use of path compression in heaps to prove small model properties of logics of linked structures, has been used before, e.g., in [8] and more recently in [32]. Previous work on parameterized systems ([1]) relies on a small model theorem for checking inductiveness of assertions. The small model property there is similar to the one here with respect to stratified data. However, with respect to unstratified data (such as graphs), [1] takes the approach of using logical instantiation as a heuristic, whereas here completeness is achieved using graph-theoretic methods.

In this paper we use the ranking abstraction method of [4] to verify liveness properties, and we make use of the ranking functions over singly-linked lists defined in [3]. In addition, we define a new ranking function over single-parent heaps and prove a small


Figure 2: Program List-REVERSAL
model property for the extended logic. For a discussion of work related to ranking abstraction independent of shape analysis, see [4] and [2].
[18] is an abstract-interpretation-based framework that combines abstract domains over sets of heap nodes, with numerical abstract domains. This results in an analysis that can verify the termination of programs in which the proof relies on relationships between cardinalities of sets of heap nodes. The approach is analogous to our method of combining shape predicates with ranking functions that measure the cardinality of sets of heap node. [13] defines a logic for reasoning about programs that manipulate pointers and data of structures with single and multiple links, as well as arrays. The logic is parameterized by the logic of the underlying data domain. The strength of the approach is in its ability to deal with combinations of data structures and arbitrary decidable data domains. However, the approach has yet to be applied to liveness problems.

## 2. Motivating Examples

Before giving the formal framework, we describe two motivating examples that manipulate the most common heap structures - (singly) linked list and tree. For both examples we assume memory of size $h$, a system's parameter, with 0 representing nil. The first example, List-Reversal, deals with a in-place list reversal. The statement in line 2 denotes a simultaneous assignment of the three variables on the left-handside to the (old) values of the expressions in the right-hand-side. The second, TreeINSERT, with insertion of a node $n$ to a sorted binary tree. If the data contained in node $n$ is not already contained in the tree, then $n$ is inserted as a new leaf. The tree is assumed to be "non-sharing" - no two distinct links lead to the same node, and "well-formed" - nil pointers lead to nil nodes without data. Otherwise the tree is not modified. A formal description of the programs are in the sequel. Figures 2 and 3 describe the programs.

For List-Reversal we show:

1. Every node reachable from the initial node remains reachable upon termination;
2. The program eventually terminates.

## For Tree-Insert we show:

1. Every node reachable from the root remains so;


Figure 3: Program Tree-Insert
2. No node, but the one inserted, that is not initially reachable from the root becomes reachable upon termination;
3. If the new node contains data not initially in the tree then the node is inserted in the tree;
4. The program eventually terminates;

We first translate the program into a formal model called "single parent heap systems" based on the [3] model of "finite heap systems" which in turn is based on the [1] model of "bounded fair discrete systems" specialized for the case of heaps where each node has a single "parent"; we later show how the model can encompass heaps where each node has several children.

We shall use the above two examples as running examples throughout the paper. At Section 7 we describe other examples.

## 3. The Formal Framework

In this section we present our computational model.

### 3.1. Fair Discrete Systems

As our computational model, we take a fair discrete system (FDS) $\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$, where

- $V$ - A set of system variables. A state of $\mathcal{D}$ provides a type-consistent interpretation of the variables $V$. For a state $s$ and a system variable $v \in V$, we denote by $s[v]$ the value assigned to $v$ by the state $s$. Let $\Sigma$ denote the set of all states over $V$.
- $\Theta$ - The initial condition: An assertion (state formula) characterizing the initial states.
- $\rho\left(V, V^{\prime}\right)$ - The transition relation: An assertion, relating the values $V$ of the variables in state $s \in \Sigma$ to the values $V^{\prime}$ in a $\mathcal{D}$-successor state $s^{\prime} \in \Sigma$. We assume that every state has a $\rho$-successor.
- $\mathcal{J}$ - A set of justice (weak fairness) requirements (assertions); A computation must include infinitely many states satisfying each of the justice requirements.
- $\mathcal{C}$ - A set of compassion (strong fairness) requirements: Each compassion requirement is a pair $\langle p, q\rangle$ of state assertions; A computation should include either only finitely many $p$-states, or infinitely many $q$-states.

For an assertion $\psi$, we say that $s \in \Sigma$ is a $\psi$-state if $s \models \psi$.
A run of an FDS $\mathcal{D}$ is a possibly infinite sequence of states $\sigma: s_{0}, s_{1}, \ldots$ satisfying the requirements:

- Initiality - $s_{0}$ is initial, i.e., $s_{0} \models \Theta$.
- Consecution - For each $\ell=0,1, \ldots$, the state $s_{\ell+1}$ is a $\mathcal{D}$-successor of $s_{\ell}$. That is, $\left\langle s_{\ell}, s_{\ell+1}\right\rangle \models \rho\left(V, V^{\prime}\right)$ where, for each $v \in V$, we interpret $v$ as $s_{\ell}[v]$ and $v^{\prime}$ as $s_{\ell+1}[v]$.

A computation of $\mathcal{D}$ is an infinite run that satisfies

- Justice - for every $J \in \mathcal{J}, \sigma$ contains infinitely many occurrences of $J$-states.
- Compassion - for every $\langle p, q\rangle \in \mathcal{C}$, either $\sigma$ contains only finitely many occurrences of $p$-states, or $\sigma$ contains infinitely many occurrences of $q$-states.

We say that a temporal property $\varphi$ is valid over $\mathcal{D}$, denoted by $\mathcal{D} \models \varphi$, if for every computation $\sigma$ of $\mathcal{D}, \sigma \models \varphi$. We are interested in safety properties, of the form $\square p$, and progress properties, of the form $\square(p \rightarrow \diamond q)$, where $p$ and $q$ are state assertions.

### 3.2. Single-Parent Heaps

A single-parent heap system (SPHS) is an extension of the model of finite heap systems of [3] specialized for representing trees.

Such a system is parameterized by a positive integer $h$, which is the heap size. Some auxiliary arrays may be used to specify more complex structures (e.g., ordered trees). However, each node $u$ has a single link to which we refer as its "parent", and denote it by parent ( $u$ ).

Let $h>0$ be the heap size. We allow the following data types:
bool Variables whose values are boolean. With no loss of generality, we assume that all finite domain (unparameterized) values are encoded as bools;
index Variables whose value is in the range $[0 . . h]$;
index $\rightarrow$ bool arrays (bool arrays) that map heap elements to boolean values. We denote the set of boolean arrays by $\mathcal{B}$;
index $\rightarrow$ index arrays (index arrays), that describe the heap structure. We allow at most a single index arrays, which we usually denote by parent for trees and by $N x t$ for lists.

We assume a signature of variables of all of these types. Constants are introduced as variables with reserved names. Thus, we admit the boolean constants FALSE and True, and the index constant 0 . In order to have all functions in the model total, we define both bool and index arrays as having the domain index. An SPHS is well-formed if it never assigns a non- 0 value to $Z[0]$ for any (bool or index) array $Z$. Formally, we say that a single-parent state is well formed if the state satisfies:

$$
\begin{equation*}
Z[0]=0 \wedge \bigwedge_{B \in \mathcal{B}}(\neg B[0]) \tag{1}
\end{equation*}
$$

where $Z$ is the single index array (i.e., parent or $N x t$ ).
On the other hand, unless stated otherwise, all quantifications are taken over the range [1.. $h$ ].

We refer to index elements as nodes. If in state $s$, the index variable $x$ has the value $\ell$, then we say that in $s, x$ points to the node $\ell$. An index term is the constant 0 , an index variable, or an expression $Z[y]$, where $Z$ is an index array and $y$ is an index variable.

Atomic formulae are defined as follows:

- If $x$ is a boolean variable, $B$ is a bool array, and $y$ is an index variable, then $x$ and $B[y]$ are atomic formulae.
- If $t_{1}$ and $t_{2}$ are index terms, then $t_{1}=t_{2}$ is an atomic formula.
- A transitive closure formula (tcf) of the form $Z^{*}\left(x_{1}, x_{2}\right)$, denoting that $x_{2}$ is $Z$-reachable from $x_{1}$, where $x_{1}$ and $x_{2}$ are index variables and $Z$ is an index array.

Preservation assertions describe the variables that are not changed by a transition. There are two forms of preservation assertions: (i) $\operatorname{pres}\left(\left\{v_{1}, \ldots, v_{k}\right\}\right)=\bigwedge_{i=1}^{k} v_{i}^{\prime}=v_{i}$ where all $v_{i}$ 's are scalar (bool or index) variables, and (ii) $\operatorname{pres}_{H}\left(\left\{a_{1}, \ldots, a_{k}\right\}\right)=$ $\bigwedge_{i=1}^{k} \bigwedge_{h \notin H} \cdot a_{i}^{\prime}[h]=a_{i}[h]$ where all $a_{i}$ 's are arrays and $H$ is a (possible empty) set of index variables, which denotes that all but finitely many (usually a none or a single) entries of arrays indexed by certain nodes remain intact. We abuse notation of preservation and use the expression $\operatorname{presEx}\left(\left\{v_{1}, \ldots, v_{k}\right\}\right)$ to denote the preservation of all variables, excluding the terms $v_{1}, \ldots, v_{k}$, which are either variables or array terms of the form $Z[x]$.

A restricted $A$-assertion is either one of the following forms: $\forall y . Z[y] \neq u$, $\forall y . Z[y] \neq u \vee B[y], \forall y . Z[y] \neq u \vee \neg B[y], \operatorname{pres}_{H}(Z)$, and $\operatorname{pres}_{H}(B)$, where $Z$ is an index array and $B$ is a bool array, and $H$ is a set of index variables. A restricted EA-assertion is a formula of the form $\exists \vec{x} \cdot \psi(\vec{u}, \vec{x})$, where $\vec{x}$ is a list of index variables, and $\psi(\vec{u}, \vec{x})$ is a boolean combination of atomic formulae and restricted A-assertions, where restricted A-assertions appear under positive polarity. Note that in restricted EA-assertions, universally quantified variables may not occur in tcf's. As the initial condition $\Theta$ we allow restricted EA-assertions, and in the transition relation $\rho$ and fairness requirements we only allow restricted EA-assertions without tcf's. Properties are safety properties, of the form $\forall \vec{x} .\left(\phi_{1}(\vec{x}) \rightarrow\right.$$\left.\phi_{2}(\vec{x})\right)$ ) where $\phi_{1}$ and $\phi_{2}$ are boolean combinations of atomic formulae, and liveness properties of the form $\forall \vec{x} . \diamond(\phi(\vec{x}))$ where $\phi$ is similarly defined.

Recall the Tree-Insert example of Section 2. As mentioned in the introduction, one can easily transform a tree into a single parent structure by reversing the directions of the links, which we are going to formalize in Section 4. An ordered SPHS is one that includes a distinguished $c t:$ index $\rightarrow[1 . . k]$ array, for some constant $k$, that denotes for each heap node its place among its siblings. This allows the subtrees of a given root node to be distinguished by their ct order. We extend the assertional language with a new type of atomic formula: For each $i \in[1 . . k]$, the formula $i$-subtree ${ }_{Z}\left(x_{1}, x_{2}\right)$ denotes that $x_{1}$ is in the $i^{t h}$ subtree of $x_{2}$, where $x_{1}$ and $x_{2}$ are index variables and $Z$ is an index array. This is formally expressed by the formula

$$
i \text { subtree }_{Z}\left(x_{1}, x_{2}\right): \exists u . Z[u]=x_{2} \wedge c t[u]=i \wedge Z^{*}\left(x_{1}, u\right)
$$

We support these predicates explicitly rather than as derived forms because, due to the transitive closure over a quantified variable, they would otherwise be outside of the assertional language allowed for abstraction predicates (see Section 5). Throughout the paper, when the index array $Z$ is apparent from the context, we use the short form $i$-subtree $\left(x_{1}, x_{2}\right)$. For example, in the context of program TrEE-InSERT of Example 2, the predicates left-subtree and right-subtree denote the left and right subtree relations among nodes of the parent array, whereas left-subtree ${ }^{\prime}$ and right-subtree ${ }^{\prime}$ denote subtree relations among nodes of the parent ${ }^{\prime}$ array.

### 3.3. Examples

We present examples of transforming program manipulating multi-linked heaps into SPHS.

## Example 1 (List Reversal).

Consider program List-Reversal of Fig. 2 in Section 2. The array Nxt describes the pointer structure. We ignore the actual data values, but they can easily be added as bool type variables.

Fig. 4 describes the SPHS corresponding to program LIST-REVERSAL. Note that all the clauses in Fig. 4 are restricted EA-assertions without tcf's. The justice requirement states that as long as the program has not terminated, its execution continues.

$$
\begin{aligned}
& V: \begin{cases}N x t: & \text { array }[0 . . h] \text { of }[0 . . h] \\
x, y: & {[0 . . h]} \\
\pi: & {[1 . .3]}\end{cases} \\
& \Theta: \quad \pi=1 \wedge y=0 \\
& \rho: \quad\left[\begin{array}{l}
\pi=1 \wedge \pi^{\prime}=3 \wedge x=0 \wedge \operatorname{presEx}(\{\pi\}) \\
\vee \\
\pi=1 \wedge \pi^{\prime}=2 \wedge x \neq 0 \wedge \operatorname{presEx}(\{\pi\}) \\
\vee \\
\pi=2 \wedge \pi^{\prime}=1 \wedge x^{\prime}=\operatorname{Nxt}[x] \wedge y^{\prime}=x \wedge \\
N x t^{\prime}[x]=y \wedge \operatorname{presEx}(\{\pi, x, y, N x t[x]\}) \\
\vee \\
\pi=3 \wedge \operatorname{presEx}(\emptyset)
\end{array}\right] \\
& \mathcal{J}: \quad\{\pi \neq 1, \pi \neq 2\} \\
& \mathcal{C}: \quad \emptyset
\end{aligned}
$$

Figure 4: SPHS for Program LIST-REVERSAL

One of the safety properties one wishes to prove is that no elements are removed from the list, i.e., that every element initially reachable from $x$ is reachable from $y$ upon termination. This property can be expressed by:

$$
\begin{equation*}
\forall t .\left(\pi=1 \wedge t \neq 0 \wedge N x t^{*}(x, t)\right) \rightarrow \square\left(\pi=3 \rightarrow N x t^{*}(y, t)\right) \tag{2}
\end{equation*}
$$

However, since here $N x t^{*}(x, t)$ is under a universal assertion, which is disallowed by our definition of restricted EA-assertion, we augment the program with a skolem constant $t$, whose initial value is unconstrained and remained fixed henceforth. The validity of Formula (2) reduces to the validity of

$$
\begin{equation*}
\left(\pi=1 \wedge t \neq 0 \wedge N x t^{*}(x, t)\right) \rightarrow \square\left(\pi=3 \rightarrow N x t^{*}(y, t)\right) \tag{3}
\end{equation*}
$$

In addition, one may want to prove the liveness property

$$
\begin{equation*}
\diamond(\pi=3) \tag{4}
\end{equation*}
$$

## Example 2 (Single Parent Encoding of Tree Insertion).

Recall the Tree-Insert program from Section 2. We present, in Fig. 5, a similar program, SP-Tree-Insert, encoded as a single-parent. In Subsection 4.3 we prove that the transformation is sound. To allow for the presentation of a sorted binary tree, we use an array $c t$ (child-type) such that $c t[u]$ equals left or right if node $u$ is, respectively, the left or right child of its parent. The initial condition captures the well-formedness of Formula (1) and the requirement that any two children of the same parent must have different child-types. (In Subsection 4.2 we term the conjunction of these two properties well_formed ${ }_{s}$.)

The $\epsilon$-expressions, $\epsilon j . \operatorname{cond}(j)$ in lines 8 and 12 denote "choose any node $j$ that satisfies cond." For both statements in this program, it is easy to see that there is

```
parent : array \([0 . . h]\) of \([0 . . h]\)
ct : array \([0 . . h]\) of \(\{\) left, right \(\}\)
data : array [0..h] of [0..k]
\(r:[0 . . h] \quad\) init \(r>0 \wedge \operatorname{parent}[r]=0\)
\(t, n:[0 . . h] \quad\) init \(t=r \wedge n>0 \wedge \operatorname{parent}[n]=0 \wedge\)
    \(\forall u \cdot \operatorname{parent}[u] \neq n\)
done : bool
init \(\quad\) done \(=\) FALSE
init \(\quad: \quad\) well_formed \(\wedge \forall i \neq j . \operatorname{parent}[i]=\operatorname{parent}[j] \neq 0 \rightarrow c t[i] \neq c t[j]\)
1: while \(\neg\) done do
```

```
\([2: \quad\) if \(\operatorname{data}[n]=\operatorname{data}[t]\) then
```

$[2: \quad$ if $\operatorname{data}[n]=\operatorname{data}[t]$ then
3:done := TRUE
3:done := TRUE
4: elseif data $[n]<d a t a[t]$ then
4: elseif data $[n]<d a t a[t]$ then
5 : if $\forall j . \operatorname{parent}[j] \neq t \vee c t[j] \neq$ left then
5 : if $\forall j . \operatorname{parent}[j] \neq t \vee c t[j] \neq$ left then
$6:($ parent $[n], c t[n]):=(t$, left $)$
$6:($ parent $[n], c t[n]):=(t$, left $)$
7 : done $:=$ TRUE
7 : done $:=$ TRUE
else
else
$8: t:=\epsilon j \cdot \operatorname{parent}[j]=t \wedge c t[j]=$ left $]$
$8: t:=\epsilon j \cdot \operatorname{parent}[j]=t \wedge c t[j]=$ left $]$
9: elseif $\forall j$.parent $[j] \neq t \vee c t[j] \neq$ right then
9: elseif $\forall j$.parent $[j] \neq t \vee c t[j] \neq$ right then
$10:($ parent $[n], c t[n]):=(t$, right $)$
$10:($ parent $[n], c t[n]):=(t$, right $)$
11 : done $:=$ TRUE
11 : done $:=$ TRUE
else
else
$12: t:=\epsilon j \cdot \operatorname{parent}[j]=t \wedge c t[j]=\operatorname{right} \quad]$
$12: t:=\epsilon j \cdot \operatorname{parent}[j]=t \wedge c t[j]=\operatorname{right} \quad]$
$13:$

```

Figure 5: Program SP-TREE-INSERT
exactly one node \(j\) that meets cond. However, this is not always the case, and then such an assignment is interpreted non-deterministically. We also allow for universal tests, as those in lines 5 and 9, that test for existence of a particular node's left or right child.

In Fig. 6 we describe the SPHS for SP-Tree-Insert. For the transition relation \(\rho\), we skip the description of the transition between lines 9 to 12 (denoting them by \(\rho_{9 . .12}\) ) which is similar to the transition between lines 5 and 8 . There we add a boolean error variable initialized to FALSE, and set whenever Formula (1) is violated.

As discussed in Section 2, one may wish to show, for example, that program in SP-TREE-INSERT no node is every lost:
\[
\begin{equation*}
\text { parent }^{*}(x, r) \rightarrow \square \text { parent }^{*}(x, r) \tag{5}
\end{equation*}
\]
and, not node (but possibly \(n\) ) is ever gained:
\[
\begin{equation*}
x \neq n \wedge \neg \text { parent }^{*}(x, r) \rightarrow \square \neg \text { parent }^{*}(x, r) \tag{6}
\end{equation*}
\]

Similarly to Example 1 we replace the universal quantification with a skolem constant \(x\).
\(V:\left\{\begin{array}{lll}\text { parent } & : & \text { array }[0 . . h] \text { of }[0 . . h] \\ c t & : & \text { array }[0 . . h] \text { of }\{\text { left }, \text { right }\} \\ d a t a & : & \text { array }[0 . . h] \text { of }[0 . . k] \\ r & : & {[0 . . h]} \\ t, n & : & {[0 . . h]} \\ \text { done } & : & \text { bool } \\ \pi & : & {[1 . .13]} \\ \text { error } & : & \text { bool }\end{array}\right.\)
    \(\neg\) parent \(^{*}(n, r) \wedge \neg\) parent \(^{*}(\) parent \([r], r) \wedge r>0 \wedge\) parent \([r]=0 \wedge\)
\(\Theta: \quad \operatorname{parent}[0]=0 \wedge \forall i \neq j .(\operatorname{parent}[i]=\operatorname{parent}[j] \neq 0 \rightarrow c t[i] \neq c t[j]) \wedge\)
    \(t=r \wedge n>0 \wedge \operatorname{parent}[n]=0 \wedge \forall u .(\) parent \([u] \neq n) \wedge \neg\) done \(\wedge\) ᄀerror
    error \(\wedge \operatorname{presEx}(\emptyset)\)
        \(\vee\)
        \(\neg\) error \(\wedge\)
            \(\pi=1 \wedge \pi^{\prime}=2 \wedge \neg\) done \(\wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=1 \wedge \pi^{\prime}=13 \wedge\) done \(\wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=2 \wedge \pi^{\prime}=3 \wedge \operatorname{data}[t]=\operatorname{data}[n] \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=2 \wedge \pi^{\prime}=4 \wedge \operatorname{data}[t] \neq \operatorname{data}[n] \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=3 \wedge \pi^{\prime}=1 \wedge\) done \(^{\prime} \wedge \operatorname{presEx}(\{\pi\), done \(\})\)
        \(\vee \pi=4 \wedge \pi^{\prime}=5 \wedge \operatorname{data}[n]<\operatorname{data}[t] \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=4 \wedge \pi^{\prime}=9 \wedge \operatorname{data}[t] \leq \operatorname{data}[n] \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=5 \wedge \pi^{\prime}=6 \wedge(\forall j\).parent \([j] \neq t \vee c t[j] \neq\) left \() \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=5 \wedge \pi^{\prime}=8 \wedge(\exists j\).parent \([j]=t \wedge c t[j]=\) left \() \wedge \operatorname{presEx}(\{\pi\})\)
        \(\vee \pi=6 \wedge n=0 \wedge\) error \(^{\prime} \wedge \operatorname{presEx}(\{\) error \(\})\)
        \(\vee \pi=6 \wedge \pi^{\prime}=7 \wedge n \neq 0 \wedge\) parent \(^{\prime}[n]=t \wedge c t^{\prime}[n]=\) left
            \(\wedge \operatorname{presEx}(\{\pi\), parent \([n], c t[n]\})\)
        \(\vee \pi=7 \wedge \pi^{\prime}=1 \wedge\) done \(^{\prime} \wedge \operatorname{presEx}(\{\pi\), done \(\})\)
        \(\vee \pi=8 \wedge \pi^{\prime}=1 \wedge\left(\exists j \cdot \operatorname{parent}[j]=t \wedge c t[j]=\right.\) left \(\left.\wedge t^{\prime}=j\right)\)
            \(\wedge \operatorname{presEx}(\{\pi, t\})\)
        \(\vee \rho_{9 . .12}\)
        \(\vee \pi=13 \wedge \pi^{\prime}=13 \wedge \operatorname{presEx}(\{\pi\})\)
\(\mathcal{J}:\left\{\begin{array}{c}\pi \neq 1, \pi \neq 2, \pi \neq 3, \pi \neq 4, \pi \neq 5, \pi \neq 6, \\ \pi \neq 7, \pi \neq 8, \pi \neq 9, \pi \neq 10, \pi \neq 11, \pi \neq 12\end{array}\right\}\)
\(\mathcal{C}: \emptyset\)

Figure 6: SPHS for program SP-TREE-INSERT

\section*{4. From Multi-Linked Heaps into Single-Parent Heaps}

We define multi-linked heap systems with a bounded out-degree on nodes, show how to transform such a system into its ordered single-parent heap counterpart, and prove the correctness of the transformation. Note that if the bounded out-degree is 1 then the heap is a single-parent and thus needs no special treatment. We therefore restrict here to out-degrees of 2 or more.

\subsection*{4.1. Multi-linked Heap Systems}

A multi-linked heap system is represented similarly to an SPHS, only, instead of having a single index arrays (parent), we allow for some \(k>1\) index arrays, each describing one of the links a node may have. We denote these arrays by \(\operatorname{lin} k_{1}, \ldots, \operatorname{lin}_{k}\). Thus, each \(\operatorname{link}_{i}\) is an array \([0 . . h] \rightarrow[0 . . h]\). As in the single-parent case, we do not allow programs to assign a non- 0 value to any \(Z[0]\) (for a bool or index array \(Z\) ), and we assume it is always 0 . We are mainly interested in non-sharing heaps, where no two distinct (non-0) links can lead into the same non-0 node. Intuitively, non-sharing captures the property that reversal of the links of a multi-linked heap system yields an SPHS. Formally, a multi-linked heap system is well-formed if the following all holds:
1. For every \(i=1, \ldots, k, \operatorname{lin}_{i}[0]=0\).
2. For every bool array \(B, \neg B[0]\).

We refer to the conjunction of the above two requirements by the formulae well_formed \(_{m}\). In addition, we may require that:
1. No two distinct positive nodes may share a common positive child. This requirement can be formalized as
\[
\forall j, \ell \in[1 . . h], i, r \in[1 . . k] .(j \neq \ell) \wedge\left(\operatorname{link}_{i}[j]=\operatorname{link}_{r}[\ell]\right) \rightarrow \operatorname{lin}_{i}[j]=0
\]
2. No two distinct links of a positive node may point to the same positive child. This can be formalized as
\[
\forall j \in[1 . . h], s, t \in[1 . . k] .(s \neq t) \wedge\left(\operatorname{link}_{s}[j]=\operatorname{link}_{t}[j]\right) \rightarrow \operatorname{link}_{s}[j]=0
\]

We refer to the conjunction of the above requirements by the formula no_sharing. A state violating no_sharing is called a sharing state. A multi-linked system is called sharing-free if none of its computations ever reaches a sharing state, nor does a computation ever attempt to assign a value to \(Z[0]\) for some array \(Z\).

Let \(\mathcal{D}:\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle\) be a \(k\)-bounded multi-linked heap system. Fig. 7 describes a BNF-like syntax of the assertions used in describing \(\mathcal{D}\), which we refer to as Massertions. As will become clear soon, M-assertions are the multi-linked equivalents of the single-parent restricted EA-assertions. In Fig. 7, Ivar denotes an unprimed index variable, Iarr denotes an unprimed index array, Bvar denotes an unprimed bool variable, and Barr denotes an unprimed bool array. The expression reach ( \(x, y\) ) abbreviates \((x, y) \in\left(\bigcup_{i=1}^{k} \operatorname{lin} k_{i}\right)^{*}\), and the expression cycle \((x)\) abbreviates \((x, x) \in\) \(\left(\bigcup_{i=1}^{k} \operatorname{lin} k_{i}\right)^{+}\). The Preservation assertion is just like in the single-parent case and we require that if Assign appears in \(\tau\), then the Preservation assertion that is conjoined with it includes preservation of all variables that don't appear in the left-hand-side of any clause of Assign.

For example, a binary tree is a multi-linked heap structure with bound 2 and nosharing. Each of left and right is a link. Program Tree-Insert in Fig. 3 of Section 2 is the standard algorithm for inserting a new node, \(n\), into a sorted binary tree rooted at \(r\). The properties listed in Example 2 can be expressed by the following two properties, which are the counterparts of Formula (5) and Formula (6).
```

MCond1 ::= TRUE | Bvar | Barr[Ivar] | Ivar=Ivar |
Ivar=0 | Iarr[Ivar]= Ivar | Iarr[Ivar]=0 |
MCond1 V MCond1 | \negMCond1
MCond2 ::= MCond1 | reach(Ivar,Ivar) | cycle(Ivar) |
\MCond2 | Mcond2 \vee Mcond2
Assign ::= \epsilon | Bvar' | \negBvar' | Barr'[Ivar] | \negBarr'[Ivar] |
Bvar'= Bvar | Ivar' = 0 | Ivar' = Ivar |
Iarr'[Ivar] = Ivar | Iarr'[Ivar] = 0 |
Assign ^Assign | Ivar' = Iarr[Ivar]
\Theta ::= MCond2 ^ no_sharing
\rho ::= TRUE | MCond1 ^ Assign ^ Preservation | \rho\vee \rho
J ::= \emptyset | J \cup{Mcond1}
\mathcal { C } : : = \emptyset \| \mathcal { C } \cup \{ ( M C o n d 1 , M C o n d 1 ) \}

```

Figure 7: Grammar for M-Assertions for Multi-Linked Systems
\[
\begin{gather*}
\forall x . \operatorname{reach}(r, x) \rightarrow \square \operatorname{reach}(r, x)  \tag{7}\\
\forall x . x \neq n \wedge \neg \operatorname{reach}(r, x) \rightarrow \square \neg \operatorname{reach}(r, x) \tag{8}
\end{gather*}
\]

Section 2 also lists another safety property for TREE-INSERT, namely, that if needs be, the new node is inserted into the tree. Formally,
\[
\begin{equation*}
(\forall u . \operatorname{reach}(r, u) \rightarrow \operatorname{data}[u] \neq \operatorname{data}[n]) \rightarrow \square(\pi=13 \rightarrow \operatorname{reach}(r, n)) \tag{9}
\end{equation*}
\]

\subsection*{4.2. Transforming Multi-Linked Heaps into Single-Parent Heap Systems}

Let \(\mathcal{D}_{m}:\left\langle\mathcal{V}_{m}, \Theta_{m}, \rho_{m}, \mathcal{J}_{m}, \mathcal{C}_{m}\right\rangle\) be a \(k\)-bounded multi-linked heap system. Thus, \(\mathcal{V}_{m}\) includes the index arrays \(\operatorname{link}_{1}, \ldots, \operatorname{link}_{k}\). We transform \(\mathcal{D}_{m}\) into an SPHS \(\mathcal{D}_{s}:\left\langle\mathcal{V}_{s}, \Theta_{s}, \rho_{s}, \mathcal{J}_{s}, \mathcal{C}_{s}\right\rangle\). Intuitively, to transform \(\mathcal{D}_{m}\) into \(\mathcal{D}_{s}\), we replace the index link arrays with a single index parent array that reverses the direction of the links, and assign to \(c t[i]\) (child type) the "birth order" of \(i\) in the heap. The variable error \(\in\) \(\mathcal{V}_{s} \backslash \mathcal{V}_{m}\) is boolean and is set when \(\mathcal{D}_{m}\) cannot be transformed into a singe-parent system. This is caused by either an assignment to \(Z[0]\) or by a violation of the nonsharing requirement. When such an error occurs, error is raised, and remains so, i.e., \(\rho_{s}\) implies error \(\rightarrow\) error \(^{\prime}\).

Formally we transform \(\mathcal{D}_{m}\) into \(\mathcal{D}_{s}\), as follows. The set of variables \(\mathcal{V}_{s}\) consists of:
1. \(\mathcal{V}_{m} \backslash\left\{\operatorname{link}_{1}, \ldots, \operatorname{link}_{k}\right\}\), i.e., we remove from \(\mathcal{V}_{m}\) all the link arrays;
2. An index array parent : \([0 . . h] \mapsto[0 . . h]\) that does not appear in \(\mathcal{V}_{m}\);
3. A bool array \(c t:[0 . . h] \mapsto[0 . . k]\) that does not appear in \(\mathcal{V}_{m}\) (recall our convention that "bool" can be any finite-domain type);
4. A new bool variable error; error is set when \(\mathcal{D}_{m}\) contains an erroneous transition such as one that introduces sharing in the heap, or one that attempts to assign values to \(Z[0]\) for some array \(Z\).

We extend the definition of the well-formedness of Subsection 3.2 and require that single parent state is well formed, and denote it by well_formed \({ }_{s}\), if:
\[
\begin{gather*}
\operatorname{parent}[0]=0 \wedge \bigwedge_{B \in \mathcal{B}}(\neg B[0]) \wedge \\
\forall i \neq j .(\text { parent }[i]=\operatorname{parent}[j] \neq 0 \rightarrow c t[i] \neq c t[j]) \tag{10}
\end{gather*}
\]

Note that the first conjunct is exactly Formula (1) of Subsection 3.2.
We next transform the m-assertions used in multi-linked heap systems into restricted EA-assertions used for the SPHS's. In fact, since no sharing and Preservation are easily translatable into restricted EA-assertions over \(\mathcal{V}_{s}\), it suffices to transform the M-asserions MCond1, MCond2, and Assign into restricted EA-assertions over \(\mathcal{V}_{s}\). Since \(\rho_{m}\) can be easily expressed as a disjunction of M-assertions, this is done by induction on the structure of the three M -assertions.

Let \(\psi\) be an M -assertion. In the following cases, \(\psi\) remains unchanged in the transformation:
1. \(\psi\) contains no reference to index variables and arrays;
2. \(\psi\) is of the form \(x_{1}=x_{2}\) where \(x_{1}\) and \(x_{2}\) are both primed, or both unprimed, index variables;
3. \(\psi\) is of the form \(x_{1}=x_{2}\) where \(x_{1}\) is a primed, and \(x_{2}\) is an unprimed, index variable;
4. \(\psi\) is of the form \(x=0\) where \(x\) is a (either primed or unprimed) index variable;
5. \(\psi\) is of the form \(B[x]\), where \(B\) is an unprimed bool array.

The other cases are treated below. We now denote primed variables explicitly, e.g., \(x_{1}\) refers to an unprimed variable, and \(x_{1}^{\prime}\) refers to a primed variable:
1. An assertion of the form \(\operatorname{lin} k_{j}\left[x_{2}\right]=x_{1}\) is transformed into
\[
\begin{array}{ll} 
& \left(x_{2}=0 \wedge x_{1}=0\right) \\
\vee \quad & \left(x_{2} \neq 0 \wedge x_{1}=0 \wedge \forall z \cdot\left(\text { parent }[z] \neq x_{2} \vee c t[z] \neq j\right)\right) \\
\vee \quad & \left(x_{2} \neq 0 \wedge x_{1} \neq 0 \wedge \operatorname{parent}\left[x_{1}\right]=x_{2} \wedge c t\left[x_{1}\right]=j\right)
\end{array}
\]

In the case that \(x_{2} \neq 0\) and \(x_{1}=0, x_{2}\) should have no \(j^{t h}\) child. If \(x_{2} \neq 0\) and \(x_{1} \neq 0\), then \(x_{1}\) should have \(x_{2}\) as a parent and the child type of \(x_{1}\) should be \(j\).
2. A transitive closure formula reach \(\left(x_{1}, x_{2}\right)\) is transformed into
\[
\left(x_{1} \neq 0 \wedge x_{2} \neq 0 \wedge \operatorname{parent}^{*}\left(x_{2}, x_{1}\right)\right) \vee\left(x_{2}=0\right)
\]

The first disjunct deals with the case where \(x_{1}\) and \(x_{2}\) are both non- 0 nodes, and then the reachability direction is reversed, reflecting reversal of heap edges in the
transformation to an SPHS. The second disjunct deals with the case that \(x_{2}=0\), and then, since \(k>1\) and sharing is not allowed, there is a path from any node into 0 .
3. A transitive closure formula \(\operatorname{cycle}(x)\), where \(x\) is an index variable, is transformed into parent* (parent \([x], x)\).
4. An assertion of the form \(x_{1}^{\prime}=\operatorname{lin} k_{j}\left[x_{2}\right]\) is transformed into:
\[
\begin{array}{ll} 
& \left(x_{2}=0 \wedge x_{1}^{\prime}=0\right) \\
\vee & \left(x_{2} \neq 0 \wedge x_{1}^{\prime}=0 \wedge \forall y \cdot\left(\text { parent }[y] \neq x_{2} \vee c t[y] \neq j\right)\right) \\
\vee & \left(x_{2} \neq 0 \wedge \exists y \cdot\left(\operatorname{parent}[y]=x_{2} \wedge c t[y]=j \wedge x_{1}^{\prime}=y\right)\right.
\end{array}
\]

In case \(x_{2}=0\), this transition sets \(x_{1}\) to 0 since we assume that in non-sharing states \(\operatorname{link}_{j}[0]=0\) for every \(j=1, \ldots, k\). Otherwise, if \(x_{2}\) has no \(j^{\text {th }}\) child, then \(x_{1}\) is set to 0 . Otherwise, there exists a \(y\) which is the \(j^{\text {th }}\) child of \(x_{2}\), and then \(x_{1}\) is set to \(y\).
5. The transformation of an assertion of the form \(B^{\prime}[x]\), where \(B\) is an unprimed bool array, depends on the polarity of \(B^{\prime}[x]\). If positive, it is transformed into:
\[
\left(x=0 \wedge \text { error }^{\prime}\right) \vee\left(x \neq 0 \wedge B^{\prime}[x]\right)
\]

The error condition reflects an attempt to assign True to \(B[0]\). If the assertion \(B^{\prime}[x]\) appears under negative polarity, then no erroneous assignment is possible, and the assertion remains unchanged by the transformation.
6. An assertion of the form \(\operatorname{link} k_{j}^{\prime}\left[x_{1}\right]=x_{2}\) is transformed into:
\[
\begin{aligned}
\text { Err } & \wedge \text { error }^{\prime} \vee \\
& \neg \text { Err }^{\prime} \\
\wedge & \left(x_{2}=0 \vee\left(x_{2} \neq 0 \wedge \text { parent }^{\prime}\left[x_{2}\right]=x_{1} \wedge \text { ct }^{\prime}\left[x_{2}\right]=j\right)\right) \\
\wedge & \binom{\forall z \cdot\left(\text { parent }[z] \neq x_{1} \vee c t[z] \neq j\right) \vee}{\exists z \cdot\left(\text { parent }[z]=x_{1} \wedge c t[z]=j \wedge\left(z=x_{2} \vee \text { parent }^{\prime}[z]=0\right)\right)}
\end{aligned}
\]

Where Err is defined by:
\[
\begin{aligned}
& \left(x_{1}=0\right. \\
\vee \quad\left(x_{2} \neq 0\right. & \left.\wedge \operatorname{parent}\left[x_{2}\right] \neq 0 \wedge\left(\operatorname{parent}\left[x_{2}\right] \neq x_{1} \vee c t\left[x_{2}\right] \neq j\right)\right)
\end{aligned}
\]
I.e., the assignment may cause an error by either attempting to assign a nonzero value to \(\operatorname{link} k_{j}[0]\), or by introducing sharing (when \(x_{2}\) either has a parent that is not \(x_{1}\), or is \(x_{1}\) 's \(i^{\text {th }}\) child for some \(i \neq j\) ).
When there is no error, \(x_{2}\) should become the \(j^{t h}\) child of \(x_{1}\) unless it is 0 , which is expressed by the first conjunct of the non-error case; in addition, any node that was the \(j^{t h}\) child of \(x_{1}\) before the transition should become "orphaned," which is expressed by the second conjunct of the non-error case.
Note that the transformation guarantees that M -assertions are transformed into restricted EA-assertions.

Example 3 (transforming TreE-INSERT).

In Example 2 we presented SP-Tree-Insert directly as an SPHS. Assume, however, that we start with TREE-INSERT, and translate its FDS into an SPHS. The FDS obtained would be very similar to that presented in Fig. 6, only that a special treatment would be required for test of the maintenance of the well-formedness. For example in statement where \(\pi=8\), in Fig. 6 the corresponding transition
\[
\pi^{\prime}=1 \wedge\left(\exists j \cdot \operatorname{parent}[j]=t \wedge c t[j]=\text { left } \wedge t^{\prime}=j\right) \wedge \operatorname{presEx}(\{\pi, t\})
\]
while had we started as we should have from an FDS of TREE-INSERT the corresponding transition, resulting from case (1) of the transformation, is
\[
\left.\begin{array}{c}
\pi^{\prime}=1 \wedge \operatorname{presEx}(\{\pi, t\}) \wedge \\
\left.\left(\begin{array}{l}
t=0 \\
\vee \quad t \neq 0 \\
\wedge t^{\prime}=0 \\
\vee \quad t \neq 0
\end{array}\right) \not t^{\prime}=0 \wedge \forall j \cdot\left(\operatorname{parent}[j]=t \wedge c t[j]=\text { left } \wedge \operatorname{parent}^{\prime}[j] \neq t \vee \operatorname{tct}=j\right]\right)
\end{array}\right)
\]

\subsection*{4.3. Correctness of Transformation}

In order for the above transformation to fit into the verification process proposed in Section 1, we have to show that the result of the verification, as carried out on the transformed system and property, holds with respect to the untransformed counterparts. Such a result is provided by Theorem 1 below. To show that the abstraction computation method of Subsection 5.3 is sound with respect to a transformed program and property, we use the translation of M -assertions into restricted EA-assertions and Theorem 2 below. For simplicity of presentation, in this section we do not take into account fairness requirements. However, it is straightforward to extend the results, i.e., show that the heap transformation preserves satisfiability of justice requirements, and that the computation transformation preserves compassion.

Let \(\mathcal{D}_{m}:\left\langle\mathcal{V}_{m}, \Theta_{m}, \rho_{m}, \mathcal{J}_{m}, \mathcal{C}_{s}\right\rangle\) be a \(k\)-bounded multi-linked heap system over the set of variables \(\mathcal{V}_{m}\), with \(k>1\), and let \(\mathcal{D}_{s}:\left\langle\mathcal{V}_{s}, \Theta_{s}, \rho_{s}, \mathcal{J}_{s}, \mathcal{C}_{s}\right\rangle\) be its transformation into an SPHS. The transformation into an SPHS induces a mapping \(\mathcal{S}: \Sigma_{m} \rightarrow \Sigma_{s}\). The mapping \(\mathcal{S}\) is formally defined below. Let \(\mathcal{S}\) be a mapping from \(\Sigma_{m}\) into \(\Sigma_{s}\), such that for every \(s_{m} \in \Sigma_{m}\), if \(s_{s}=\mathcal{S}\left(s_{m}\right)\), then the following all hold:
1. For every bool variable \(v \in \mathcal{V}_{m}, s_{s}[v]=s_{m}[v]\);
2. For every bool array \(B \in \mathcal{V}_{m}\) and \(x \in[0 . . h], s_{s}[B](x)=s_{m}[B](x)\);
3. For every index variable \(x \in \mathcal{V}_{m}, s_{s}[x]=s_{m}[x]\)
4. \(s_{s}[\) parent \(](0)=0\) and \(s_{s}[c t](0)=1\).
5. Let \(y \in[1 . . h]\). If for all \(z \in[1 . . h]\) and \(i \in[1 . . k], s_{m}\left[\operatorname{link}_{i}\right](z) \neq y\), then \(s_{s}[\) parent \(](y)=0\) and \(s_{s}[c t](y)=1\). Otherwise, \(s_{s}[\) parent \(](y)=x\) and \(s_{s}[c t](y)=j\) where \((x, j)\) is the lexicographically minimal pair in \(\{(z, i)\) : \(z \in[1 . . h], i \in[1 . . k]\), and \(\left.s_{m}\left[\operatorname{link}_{i}\right](z)=y\right\}\).
6. \(s_{m}[\) error \(]= \begin{cases}\text { FALSE, } & \text { if } s_{m} \models \text { no_sharing } \wedge \text { well_formed }_{m} \\ \text { TRUE, } & \text { otherwise }\end{cases}\)

The following observation is an immediate consequence of the definition of the mapping:

Observation 1. The inverse \(\mathcal{S}^{-1}\) is well defined for any well formed non-error state \(s_{s} \in \Sigma_{s}\). That is, if \(s_{s} \models\) well_formed \(_{s} \wedge\) नerror then there exists a state \(s_{m} \in \Sigma_{k}\) such that \(\mathcal{S}\left(s_{m}\right)=s_{s}\).

In Appendix A we prove the main theorems that establish the correctness of the transformation. Theorem 1 establishes the soundness of the transformation, that is, that every temporal property holds over the multi-linked systems iff it holds over the SPHS:

Theorem 1 (Soundness). Assume that for every reachable \(\mathcal{D}_{m}\)-state \(s_{m} \in \Sigma_{m}, s \models\) no_sharing \(\wedge\) well_formed \(_{m}\). Let \(\varphi_{m}\) be a temporal property over M-restricted \(A\) assertions over \(\mathcal{V}_{m}\), and let \(\varphi_{s}\) be \(\varphi_{m}\), where every assertion over \(\mathcal{V}_{m}\) is replaced with its transformation into a restricted EA-assertion over \(\mathcal{V}_{s}\). Then: \(\mathcal{D}_{s} \models \varphi_{s} \Longleftrightarrow \mathcal{D}_{m} \models\) \(\varphi_{m}\)

While Theorem 1 shows that validity of temporal formulae carries from multi-linked systems into single-parent ones only when the former satisfy non-sharing and wellformedness, we prove that if the latter never reaches an error state, then the former never violates non-sharing and well-formedness:

\section*{Theorem 2 (Non-sharing).}


Thus, to verify \(\mathcal{D}_{m} \models \varphi_{m}\), one would initially perform a "sanity check" by verifying \(\mathcal{D}_{s} \models \square \neg\) error. If this is successful, then the process outlined in Section 1 can be carried out. Theorem 1 guarantees not only that correctness of \(\mathcal{D}_{s}\) implies correctness of \(\mathcal{D}_{m}\), but also that a counterexample over \(\mathcal{D}_{s}\) is mappable back into \(\mathcal{D}_{m}\).

\section*{5. Abstraction}

We fix an sphs \(S=\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle\) whose set of states is \(\Sigma\) for this section.

\subsection*{5.1. Finitary Abstraction}

The material here is an overview of (a somewhat simplified version of) [22]. See there for details.

An abstraction is a mapping \(\alpha: \Sigma \rightarrow \Sigma_{A}\) for some set \(\Sigma_{A}\) of abstract states. The abstraction \(\alpha\) is finitary if the set of abstract states \(\Sigma_{A}\) is finite. We focus on abstractions that can be represented by a set of equations of the form \(u_{i}=E_{i}(V), i=1, \ldots, n\), where the \(E_{i}\) 's are assertions over the concrete variables \((V)\) and \(\left\{u_{1}, \ldots, u_{n}\right\}\) is the set of abstract variables, denoted by \(V_{A}\). Alternatively, such \(\alpha\) can be expressed by:
\[
V_{A}=\mathcal{E}_{\alpha}(V)
\]

For an assertion \(p(V)\), we define its abstraction by:
\[
\alpha(p): \quad \exists V \cdot\left(V_{A}=\mathcal{E}_{A}(V) \wedge p(V)\right)
\]

The semantics of \(\alpha(p)\) is \(\|\alpha(p)\|=\{\alpha(s) \mid s \in\|p\|\}\). Note that \(\|\alpha(p)\|\) is, in general, an over-approximation - an abstract state is in \(\|\alpha(p)\|\) iff there exists some concrete \(p\)-state that is abstracted into it. An assertion \(p\left(V, V^{\prime}\right)\) over both primed and unprimed variables is abstracted by:
\[
\alpha(p): \exists V, V^{\prime} \cdot\left(V_{A}=\mathcal{E}_{A}(V) \wedge V_{A}^{\prime}=\mathcal{E}_{A}\left(V^{\prime}\right) \wedge p\left(V, V^{\prime}\right)\right)
\]

The assertion \(p\) is said to be precise with respect to the abstraction \(\alpha\) if \(\|p\|=\) \(\alpha^{-1}(\|\alpha(p)\|)\), i.e., if two concrete states are abstracted into the same abstract state, they are either both \(p\)-states, or they are both \(\neg p\)-states. For a temporal formula \(\psi\) in positive normal form (where negation is applied only to state assertions), \(\psi^{\alpha}\) is the formula obtained by replacing every maximal state sub-formula \(p\) in \(\psi\) by \(\alpha(p)\). The formula \(\psi\) is said to be precise with respect to \(\alpha\) if each of its maximal state subformulas are precise with respect to \(\alpha\).

We restrict the abstraction predicates to be boolean combinations of atomic formulae and non-preservation universal formulae. This last restriction ensures that the language of abstraction predicates is closed under negation, an assumption needed during abstraction computation. In all cases discussed in this paper, the formulae are precise with respect to the relevant abstractions. Hence, we can restrict to the overapproximation semantics.

The \(\alpha\)-abstracted version of \(S\) is the system
\[
S^{\alpha}=\left\langle V_{A}, \alpha(\Theta), \alpha(\rho), \bigcup_{J \in \mathcal{J}} \alpha(J), \bigcup_{(p, q) \in \mathcal{C}}(\alpha(p), \alpha(q))\right\rangle
\]

From [22] we derive the soundness of finitary abstraction:
Theorem 3. For a system \(S\), abstraction \(\alpha\), and a positive normal form temporal formula \(\psi\) :
\[
S^{\alpha} \models \psi^{\alpha} \quad \Longrightarrow \quad S \models \psi
\]

Thus, if an abstract system satisfies an abstract property, then the concrete system satisfies the concrete property.

\subsection*{5.2. Predicate Abstraction}

Predicate abstraction is an instance of finitary abstraction where the abstract variables are boolean. Following [30], an initial predicate abstraction is chosen as follows: Let \(\mathcal{P}\) be the (finite) set of atomic state formulas occurring in \(\rho, \Theta, \mathcal{J}, \mathcal{C}\) and the concrete formula \(\psi\) that refer to non-control and non-primed variables. Then the abstraction \(\alpha\) is the set of equations \(\left\{B_{p}=p: p \in \mathcal{P}\right\}\). The formula \(\psi^{\alpha}\) is then checked over \(S^{\alpha}\) producing either a confirmation that \(S^{\alpha} \models \psi^{a}\) or a counterexample. In the former case, the process terminates concluding that \(S \models \psi\). Else, the counterexample produced is concreticized and checked whether it is indeed a feasible \(S\)-trace. If so,


Figure 8: Program Abstract-List-Reversal
the process terminates concluding that \(S \not \vDash \psi\). Otherwise, the concrete trace implies a refinement \(\alpha^{\prime}\) of \(\alpha\) under which the abstract error trace is infeasible. The process repeats (with \(a^{\prime}\) ) until it succeeds \(-\psi\) is proven to be valid or invalid - or the refinement reaches a fixpoint, in which case the process fails. See [15, 6, 7] for discussion of the iterated abstraction refinement method.

We demonstrate the process of predicate abstraction on program LIST-REVERSAL. In the next section we show how to automatically compute the abstraction.

\section*{Example 4 (List Reversal Abstraction).}

Consider program List-REVERSAL of Example 1 and the no-loss property Formula (3) there. To prove the safety property of Formula (3), the set \(\mathcal{P}\) consists of \(x=0, t=0\), \(N x t^{*}(x, t)\), and \(N x t^{*}(y, t)\), which we denote as the abstract variables \(x_{-} 0, t_{-} 0, r_{-} x t\), and \(r_{-} y t\) respectively.

The abstract program is Abstract-List-Reversal, shown in Fig. 8, and the abstract property corresponding to Formula (3) is:
\[
\psi^{\alpha}:\left(\Pi=1 \wedge \neg t_{-} 0 \wedge r_{-} x t\right) \rightarrow \square\left(\Pi=3 \rightarrow r_{-} y t\right)
\]
where \(\Pi\) is the program counter of the abstract program.
It is now left to check whether \(S^{\alpha} \models \psi^{\alpha}\), which can be done, e.g., using a model checker. Here, the initial abstraction is precise enough, and program ABSTRACT-LISTREVERSAL satisfies \(\psi^{\alpha}\).

\subsection*{5.3. Computing Symbolic Abstractions of Single-Parent Heap Systems}

We show how to symbolically compute the abstraction of an SPHS by extending the methodology of [3]. That methodology is based on a small model property establishing that satisfiability of a restricted assertion is checkable on a small instantiation of a system. The main effort here is dealing with the extensions to the assertional language introduced for SPHS's. For simplicity, it is assumed that all scalar values are represented by multiple boolean values.

Assume a vocabulary \(\mathcal{V}\) of typed variables, as well as the primed version of said variables. Furthermore, assume that there is a single unprimed index array in \(\mathcal{V}\) as well as a single primed one. These will be denoted throughout the rest of this section by parent and parent \({ }^{\prime}\), respectively. A model \(M\) of size \(h+1\) for \(\mathcal{V}\) consists of:
- A positive integer \(h>0\);
- For each boolean variable \(b \in \mathcal{V}\), a boolean value \(M[b] \in\{\) False, True \(\}\). It is required that \(M\) [FALSE \(]=\) FALSE and \(M[\) TRUE \(]=\) TRUE;
- For each index variable \(x \in \mathcal{V}\), a value \(M[x] \in[0 . . h]\). It is required that \(M[0]=\) 0 and \(M[H]=h\);
- For each bool array \(B \in \mathcal{V}\), a function \(M[B]:[0 . . h] \rightarrow\{\) False, True \(\}\);
- For each index array \(Z \in\{\) parent, parent' \(\}\), a function \(M[Z]:[0 . . h] \rightarrow[0 . . h]\).

Let \(\varphi\) be a restricted EA-assertion, which we fix for this section. We require that if a term of the form parent \({ }^{\prime}[u]\) occurs in \(\varphi\) where \(u\) is a free or existentially quantified variable in \(\varphi\), then \(\varphi\) also contains the preservation formula associated with parent. Note that this requirement is satisfied by any reasonable \(\varphi\) - assertions that contain primed variables occur only in proofs for abstraction computation (rather than in properties of systems), and are generated automatically by the proof system. In such cases, the assertion generated includes also the transition relation, which includes all preservation formulae. We include this requirement explicitly since the proof of the small model theorem depends on it.

Given a model \(M\), one can evaluate the formula \(\varphi\) over the model \(M\). The model \(M\) is a satisfying model for \(\varphi\), if \(\varphi\) evaluates to True in \(M\), i.e., if \(M \models \varphi\). An index term \(t \in\{u, Z[u]\}\) in \(\varphi\), where \(u\) is an existentially quantified or a free variable, is called a free term. Let \(\mathcal{T}_{\varphi}\) denote the minimal set consisting of the following:
- The term 0 and all free terms in \(\varphi\);
- For every array \(Z \in \mathcal{V}\), if \(Z[u] \in \mathcal{T}_{\varphi}\) then \(u \in \mathcal{T}_{\varphi}\);
- For every bool array \(B \in \mathcal{V}\), if \(B[u] \in \varphi\), then if \(B\) is unprimed, parent \([u] \in \mathcal{T}_{\varphi}\), and if \(B\) is primed, parent \(t^{\prime}[u] \in \mathcal{T}_{\varphi}\);
- If parent \({ }^{\prime}[u] \in \mathcal{T}_{\varphi}\) then parent \([u] \in \mathcal{T}_{\varphi}\) (this is similar to history closure of Subsection 6.2).

Let \(M\) be a model that satisfies \(\varphi\) with size greater then \(\left|\mathcal{T}_{\varphi}\right|+1\) as follows: Let \(N\) be the set of \([0 . . h]\) values that \(M\) assigns to free terms in \(\mathcal{T}_{\varphi}\). Assume that \(N=\left\{n_{0}, \ldots, n_{m}\right\}\) where \(0=n_{0}<\cdots<n_{m}\). Obviously, \(m \leq\left|\mathcal{T}_{\varphi}\right|\). Define a mapping \(\gamma: N \rightarrow[0 . . m]\) such that \(\gamma(u)=i\) iff \(M[u]=n_{i}\) (Recall that \(M\left[\mathcal{T}_{\varphi}\right]=N\), so that \(\gamma\) is onto).

We now define the model \(\bar{M}\). We start with its size and the interpretation of the scalars: \(\bar{M}[h]=m+1\); For each bool variable \(b, \bar{M}[b]=M[b]\); For each term \(u \in \mathcal{T}_{\varphi}\) \(\bar{M}[u]=\gamma(u)\).

Let \(Z \in\{\) parent, parent' \(\}\) be an index array, and let \(j \in[0 . . m]\). Consider the \(Z\)-chain in \(M \alpha: n_{j}=u_{0}, \ldots\) such that for every \(i \geq 1, M[Z]\left(u_{i-1}\right)=M\left[u_{i}\right]\). If
there is some \(i \geq 1\) such that \(u_{i} \in N\), then let \(k\) be the minimal such \(i\). We then say that \(u_{k-1}\) is the \(M\) representative of \(Z\) for \(j\) and define \(\bar{M}[Z](j)=\gamma\left(u_{k}\right)\). If no such \(i\) exists, then \(\bar{M}[Z](j)=m+1\).

As for the interpretation of \(\bar{M}\) over bool arrays, we distinguish between the case of unprimed and primed arrays. For an unprimed (resp. primed) bool array \(B\), for every \(j \in[0 . . m]\), if the \(M\) representative of parent (resp. parent') is defined and equals \(v\), then let \(\bar{M}[B](j)=M[B](v)\). Otherwise, \(\bar{M}[B](j)=M[B]\left(n_{j}\right)\). As for \(\bar{M}[B](m+1)\), let \(d \in[0 . . h]\) be the minimal such that \(M[d] \notin N\). Then \(\bar{M}[B](m+1)\) is defined to be \(M[B](d)\).

Note that \(n_{j}\) in the reduced model inherits the properties of its representative in the large model.

Example 5 (Model Reduction).


Figure 9: Model Reduction
Let parent and data be index and bool arrays respectively, and let \(\varphi\) be the assertion:
\[
\varphi: \exists u, v . u \neq v \wedge \forall y .(\operatorname{parent}[y] \neq u \vee \operatorname{data}[y])
\]

Since there are no free variables in \(\varphi\), and since no array term refers to the \(u^{t h}\) or \(v^{t h}\) element, it follows that \(\mathcal{T}_{\varphi}\) consists only of the index terms \(u\) and \(v\). Let \(M\) be a model of \(\varphi\) of size 7, as shown in Fig. 9(a). The interpretations by \(M\) of terms in \(\mathcal{T}_{\varphi}\) are the highlighted nodes. Each node \(y\) is annotated with the value \(M[\) data \(](y)\) (e.g., the node pointed to by \(u\) has data value FALSE). \(\bar{M}\), which is the reduction of \(M\) with respect to \(\mathcal{T}_{\varphi}\), is given in Fig. 9 (b). The \(M\) representative of parent for \(M[v]\) is given by the node highlighted by a dashed line in Fig. 9(a). As shown here, the node pointed to by \(v\) in \(\bar{M}\) takes on the properties of this representative node.

In Appendix B we prove:
Theorem 4 (Small Model Property). If \(M \models \varphi\) then \(\varphi\) is satisfiable by a model of size at most \(\left|\mathcal{T}_{\varphi}\right|+1\).

Given a restricted EA-assertion \(\varphi\) and a positive integer \(h_{0}\), we define the \(h_{0}\) bounded version of \(\varphi\), denoted \(\lfloor\varphi\rfloor_{h_{0}}\), to be the conjunction \(\varphi \wedge\left(H \leq h_{0}\right)\). Theorem 4 can be interpreted as stating that \(\varphi\) is satisfiable iff \(\lfloor\varphi\rfloor_{|\mathcal{T}|}\) is satisfiable.

Next, we would like to extend the small model property to the computation of abstractions. Consider first the case of a restricted EA-assertion \(\varphi\) which only refers to unprimed variables. As explained in Subsection 5.1, the abstraction of \(\varphi\) is given by \(\alpha(\varphi)=\exists V\left(V_{A}=\mathcal{E}_{A}(V) \wedge \varphi(V)\right)\). Assume that the set of (finitely many combinations of) values of the abstract system variables \(V_{A}\) is \(\left\{U_{1}, \ldots, U_{k}\right\}\). Let \(\operatorname{sat}(\varphi)\) be the subset of indices \(i \in[1 . . k]\), such that \(U_{i}=\mathcal{E}_{\alpha}(V) \wedge \varphi(V)\) is satisfiable. Then, it is obvious that the abstraction \(\alpha(\varphi)\) can be expanded into
\[
\begin{equation*}
\alpha(\varphi)\left(V_{A}\right)=\bigvee_{i \in \operatorname{Sat}(\varphi)}\left(V_{A}=U_{i}\right) \tag{11}
\end{equation*}
\]

Next, let us consider the abstraction of \(\lfloor\varphi\rfloor_{|\mathcal{T}|}\), where \(\mathcal{T}\) consists of all free terms in \(\varphi\) and \(\mathcal{E}_{\alpha}(V)\) and the variable \(H\), i.e. all the free terms in the assertion \(U_{i}=\mathcal{E}_{\alpha}(V) \wedge\) \(\varphi(V) \wedge\left(H \leq h_{0}\right)\). Our reinterpretation of Theorem 4 implies that \(\operatorname{sat}\left(\lfloor\varphi\rfloor_{|\mathcal{T}|}\right)=\) \(\operatorname{sat}(\varphi)\) which leads to the following theorem:

Theorem 5. Let \(\varphi\) be an assertion which only refers to unprimed variables, \(\alpha: V_{A}=\) \(\mathcal{E}_{A}(V)\) be an abstraction mapping, \(\mathcal{T}\) be the set of free terms in the formula \(\left(U_{i}=\right.\) \(\left.\mathcal{E}_{A}(V)\right) \wedge \varphi(V) \wedge\left(H \leq h_{0}\right)\), and \(h_{0}=|\mathcal{T}|\). Then
\[
\alpha(\varphi)\left(V_{A}\right) \quad \sim \alpha\left(\lfloor\varphi\rfloor_{h_{0}}\right)\left(V_{A}\right)
\]

Theorem 5 deals with assertions that do not refer to primed variables. It can be extended to the abstraction of an assertion such as the transition relation \(\rho\). Recall that the abstraction of such an assertion involves a double application of the abstraction mapping, an unprimed version and a primed version. Thus, we need to consider the set of free terms in the formula \(\left(U_{i}=\mathcal{E}_{A}(V)\right) \wedge U_{2}=\mathcal{E}_{A}\left(V^{\prime}\right) \wedge \rho\left(V, V^{\prime}\right)\) plus the variable \(H\).

Next we generalize these results to entire systems. For an sPHS \(S=\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle\) and positive integer \(h_{0}\), we define the \(h_{0}\)-bounded version of \(S\), denoted \(\lfloor S\rfloor_{h_{0}}\), as \(\left\langle V \cup\{H\},\lfloor\rho\rfloor_{h_{0}},\lfloor\mathcal{J}\rfloor_{h_{0}},\lfloor\mathcal{C}\rfloor_{h_{0}}\right\rangle\), where \(\lfloor\mathcal{J}\rfloor_{h_{0}}=\left\{\lfloor J\rfloor_{h_{0}} \mid J \in \mathcal{J}\right\}\) and \(\lfloor\mathcal{C}\rfloor_{h_{0}}=\) \(\left\{\left(\lfloor p\rfloor_{h_{0}},\lfloor q\rfloor_{h_{0}}\right) \mid(p, q) \in \mathcal{C}\right\}\). Let \(h_{0}\) be the maximum size of the sets of free terms for all the abstraction formulas necessary for computing the abstraction of all the components of \(S\). Then we have the following theorem:

Theorem 6. Let \(S\) be an SPHS, \(\alpha\) be an abstraction mapping, and \(h_{0}\) the maximal size of the relevant sets of free terms as described above. Then the abstract system \(S^{\alpha}\) is equivalent to the abstract system \(\lfloor S\rfloor_{h_{0}}^{\alpha}\).

We use BDD techniques to compute the abstract system \(\lfloor S\rfloor_{h_{0}}^{\alpha}\). The only manual step in the process is the choice of the state predicates. As discussed in Subsection 5.2, the initial choice is usually straightforward. One of the attractive advantages of using a model checker for the abstraction is that it can be invisible - thus, the abstraction, and checking of the (abstract) property over it, can be done completely automatically, and the user need not see the abstract program, giving rise to the method of invisible abstraction. However, because of the need for refinement, the user may actually prefer to view the abstract program.

Example 6 (List reversal-safety).

Consider again program List-Reversal. In Example 4 (of Subsection 5.2) we described its abstraction, which was manually derived, and the safety property \(\psi^{\alpha}\). In order to obtain an automatic abstraction for the system whose set of free terms is \(\mathcal{T}=\left\{0, H, x, y, t, x^{\prime}, y^{\prime}, N x t^{\prime}[x]\right\}\), we bounded the system by \(h_{0}=8\).

We compute the abstraction by initially preparing an input file describing the concrete truncated system. We then use BDD-techniques for dynamically constructing and updating a model to construct the abstract system by separately computing the abstraction of the concrete initial condition, transition relation, and fairness requirements.

Having computed the abstract system, we check the safety property \(\psi^{\alpha}\), which, of course, holds.

As a consequence, in order to compute the abstract system \(S^{\alpha}\), we can instantiate the system \(S\) to a heap of size \(h_{0}\), and use propositional methods, e.g., BDD-techniques to compute the abstract system \(\lfloor S\rfloor_{h_{0}}^{\alpha}\). Note that \(h_{0}\) is linear in the number of system variables. This process is fully automatic once the predicate base is given. The exact manner by which predicates themselves are derived (e.g., by user input or as part of a refinement loop) is orthogonal to the method presented here.

We conclude this section by presenting the abstraction and verification of safety property of SP-TREE-INSERT:

Example 7 (Tree Insert-safety).
Recall SP-TREE-INSERT from Subsection 4.1 and its properties in Formulae (7), (8), and (9):
\[
\begin{gather*}
\operatorname{reach}(r, x) \rightarrow \square \operatorname{reach}(r, x)  \tag{7}\\
x \neq n \wedge \neg \operatorname{reach}(r, x) \rightarrow \square \neg \operatorname{reach}(r, x)  \tag{8}\\
(\forall u \cdot \operatorname{reach}(r, u) \rightarrow \operatorname{data}[u] \neq \operatorname{data}[n]) \rightarrow \square(\pi=13 \rightarrow \operatorname{reach}(r, n)) \tag{9}
\end{gather*}
\]

We begin by eliminating the universal quantifiers in the no-loss and no-gain properties by introducing a skolem constant \(x\). This is done by augmenting the program with a variable with an undetermined initial value that stays constant throughout a computation. This is a purely syntactic transformation.

As for the insertion property, unfortunately the abstraction computation method disallows any occurrence of reach predicates under universal quantification. Therefore, we heuristically instantiate the universal variable \(u\) to derive the following (stronger) property:
\[
\begin{equation*}
\left(\bigwedge_{u \in\{r, n, t\}} \operatorname{reach}(r, u) \rightarrow \operatorname{data}[u] \neq \operatorname{data}[n]\right) \rightarrow \square(\pi=13 \rightarrow \operatorname{reach}(r, n)) \tag{12}
\end{equation*}
\]

We now apply predicate abstraction. We use the predicate base given by the fol-
lowing set of assertions:
\[
\mathcal{P}:\left\{\begin{array}{ll}
p_{1}: & \forall j \cdot \operatorname{parent}[j] \neq n, \\
p_{2}: & \text { left-subtree }(n, r), \\
p_{3}: & \text { right-subtree }(n, r), \\
p_{4}: & \text { parent }^{*}(t, r), \\
p_{5}: & \exists j \cdot \operatorname{parent}[j]=t,_{p_{6}:} \\
\text { data }[t]=\text { data }[n], \\
p_{7}: & \text { parent }^{*}(x, r)
\end{array}\right\}
\]

Note that the predicate \(p_{1}\) is in fact an inductive invariant, a fact that can be decided (without the use of abstraction) by directly applying Theorem 4 to check validity of the verification conditions
\[
\begin{array}{ll}
\text { I1. } & \Theta \rightarrow p_{1} \\
\text { I2. } & p_{1} \wedge \rho \rightarrow p_{1}^{\prime}
\end{array}
\]

Having decided the invariance of \(p_{1}\), it is possible to optimize the abstraction computation by removing \(p_{1}\) from the predicate base, and by constraining the concrete state space to \(p_{1}\)-states only.

\section*{6. Liveness}

\subsection*{6.1. Transition Abstraction}

State abstraction often does not suffice to verify liveness properties and needs to be augmented with transition abstraction. Let \((\mathcal{D}, \succ)\) be a partially ordered well founded domain, and assume a ranking function \(\delta: \Sigma \rightarrow \mathcal{D}\). Define a function decrease \({ }_{\delta}\) by:
\[
\text { decrease }_{\delta}=\left\{\begin{aligned}
1 & \delta \succ \delta^{\prime} \\
0 & \delta=\delta^{\prime} \\
-1 & \text { otherwise }
\end{aligned}\right.
\]

Transition abstraction can be incorporated into a system by (synchronously) composing the system with a progress monitor [22], shown in Fig. 10. The compassion require-


Figure 10: Progress Monitor \(M(\delta)\) for a Ranking \(\delta\)
ment states that if dec is 1 infinitely many times, it should also be -1 infinitely many


Figure 11: Program Nested-Loops
times. This corresponds to the well-foundedness of \((\mathcal{D}, \succ)\) : the ranking cannot decrease infinitely many times without increasing infinitely many times. To incorporate this in a state abstraction \(\alpha\), we add the defining equation \(\operatorname{dec}_{A}=\operatorname{dec}\) to \(\alpha\).

\section*{Example 8 (List Reversal Termination).}

Consider program LIST-REVERSAL and the termination property \(\diamond(\pi=3)\). The loop condition \(x \neq 0\) in line 1 implies that the set of nodes starting with \(x\) is a measure of progress. This suggests the ranking \(\delta=\left\{i \mid N x t^{*}(x, i)\right\}\) over the well founded domain \(\left(2^{\mathbb{N}}, \supset\right)\). That is, the rank of a state is the set of all nodes which are currently reachable from \(x\). As the computation progresses, this set loses more and more of its members until it becomes empty. Using a sufficiently precise state abstraction, one can model check that the abstract property \(\diamond(\Pi=3)\) indeed holds over the program.

Just like the case of predicate abstraction, we lose nothing (except efficiency) by adding potentially redundant rankings. The main advantage here over direct use of ranking functions within deductive verification is that one may contribute as many elementary ranking functions as one wishes. Assuming a finitary abstraction, it is then left to the model-checker to sort out their interaction and relevance. To illustrate this, consider the program Nested-Loops in Fig. 11. The statements \(x:=?, y:=\) ? in lines 1 and 3 denote assignments of a random natural to \(x\) and \(y\). Due to this unbounded non-determinism, a deductive termination proof of this program needs to use a ranking function ranging over lexicographic triplets, whose core is \((\pi=1, x, y)\). With augmentation, however, one need only provide the rankings \(\delta_{1}: x\) and \(\delta_{2}: y\).

\subsection*{6.2. Computing the Augmented Abstraction}

We aim to apply symbolic abstraction computation of Subsection 5.3 to systems augmented with progress monitors. However, since progress monitors are not limited to restricted A-assertions, such systems are not necessarily SPHS's. Thus, for any ranking function \(\delta\), one must show that Theorem 6 is applicable to such an extended form of SPHS's. Since all assertions in the definition of an augmented system, with the exception of the transition relation, are restricted A-assertions, we need only consider
the augmented transition relation \(\rho \wedge \rho_{\delta}\), where \(\rho\) is the unaugmented transition relation and \(\rho_{\delta}\) is defined as dec \(^{\prime}=\) decrease \(_{\delta}\). Let \(\mathcal{T}\) be a set consisting of all free terms in the assertions \(\rho \wedge \rho_{\delta}, \mathcal{E}_{\alpha}(V)\), and \(\mathcal{E}_{\alpha}\left(V^{\prime}\right)\), as well as the variable \(H\). Then Theorem 6 holds if it is the case that
\[
\begin{equation*}
\operatorname{sat}\left(\left\lfloor\rho \wedge \rho_{\delta}\right\rfloor_{|\mathcal{T}|}\right)=\operatorname{sat}\left(\rho \wedge \rho_{\delta}\right) \tag{13}
\end{equation*}
\]

Since proving Formula (13) for an arbitrary ranking is potentially a significant manual effort, we specifically consider the following commonly used ranking functions over the well founded domain \(\left(2^{\mathbb{N}}, \supset\right)\) :
\[
\begin{gather*}
\delta_{1}(x)=\left\{i \mid N x t^{*}(x, i)\right\}  \tag{14}\\
\delta_{2}(x)=\left\{i \mid N x t^{*}(i, x)\right\}  \tag{15}\\
\delta_{3}(x, y)=\left\{i \mid N x t^{*}(x, i) \wedge N x t^{*}(i, y)\right\} \tag{16}
\end{gather*}
\]

In the above, \(x, y\) are index variables, and \(N x t\) is an index array. Ranking \(\delta_{1}\) is used to measure the progress of a forward moving pointer \(x\), while ranking \(\delta_{2}\) is used to measure the progress of a backward moving pointer \(x\). Ranking \(\delta_{3}\) is used to measure the progress of pointers \(x\) and \(y\) toward each other. Throughout the rest of this section we assume that the variables \(x\) and \(y\) appearing in \(\delta_{1}, \delta_{2}\), or \(\delta_{3}\) are free terms in the unaugmented transition relation.

In order to extend the small model property to cover transition relations of the form \(\rho_{\delta}\) we impose stronger conditions on the set of terms \(\mathcal{T}\). A term set \(\mathcal{T}\) is said to be history closed if for every term of the form \(N x t[x], N x t^{\prime}[x] \in \mathcal{T}\) only if \(N x t[x] \in \mathcal{T}\). From now on, we restrict to history-closed term sets. Note that history closure implies a stronger notion of uniformity as follows: For any model \(M\) and nodes \(k, k_{1}, k_{2}\), if \(M[N x t](k)=k_{1} \neq k_{2}=M\left[N x t^{\prime}\right](k)\), then all of \(k, k_{1}, k_{2}\) are pointed to by terms in \(\mathcal{T}\).

The following theorem, whose proof is in Appendix C, establishes the soundness of our method for proving liveness for the three ranking functions we consider.

Theorem 7. Let \(S\) be an unaugmented SPHS with transition relation \(\rho\), \(\delta_{i}\) be a ranking with \(i \in\{1,2,3\}, M\) be a uniform model satisfying \(\rho \wedge \rho_{\delta}, \mathcal{T}\) be a history-closed term set containing the variable \(H\) and the free index terms in the assertions \(\rho \wedge \rho_{\delta}\), \(\mathcal{E}_{\alpha}(V)\), and \(\mathcal{E}_{\alpha}\left(V^{\prime}\right)\), and \(\bar{M}\) be the appropriate reduced model of size \(h_{0}=|\mathcal{T}|\).

Then \(\bar{M} \models \rho_{\delta_{i}}\) only if \(M \models \rho_{\delta_{i}}\).

\section*{Example 9 (List Reversal Termination).}

In Example 8 we propose ranking \(\delta_{1}\) to verify termination of program LIST-REVERSAL. From Theorem 7 it follows that there is a small model property for the augmented program. The bound of the truncated system, according to Theorem 6, is
\[
h_{0}=|\mathcal{T}|=\left|\left\{H, 0, x, y, x^{\prime}, y^{\prime}, N x t^{\prime}[x], N x t[x]\right\}\right|=8
\]

We have computed the abstraction, and proved termination of LIST-REVERSAL.


Figure 12: Program Bubble Sort

\section*{Example 10 (Tree-Insert Termination).}

Consider program TREE-INSERT and the termination property \(\diamond(\pi=13)\). The loop condition \(\neg\) done in line 1 implies that the set of nodes which decendence of \(t\) is a measure of progress. This suggests the ranking \(\delta_{2}\). That is, the rank of a state is the set of all nodes which can reachable \(x\). As the computation progresses, this set loses more and more of its members until it becomes empty. Using a sufficiently precise state abstraction, one can model check that the abstract property \(\diamond(\Pi=13)\) indeed holds over the program.

From Theorem 7 it follows that there is a small model property for the augmented program. The bound of the truncated system, according to Theorem 6 , is
\[
h_{0}=|\mathcal{T}|=\left|\left\{H, 0, r, n, t, t^{\prime}\right\}\right|=6
\]

We have computed the abstraction, and proved termination of Tree-Insert.

\section*{7. Examples}

\subsection*{7.1. Bubble Sort}

We present our experience in verifying a bubble sort algorithm on acyclic, singlylinked lists. The program is given in Fig. 12. The requirement of acyclicity is expressed in the initial condition \(N x t^{*}(x, 0)\) on the array \(N x t\). We first summarize the proof of some safety properties. We then discuss issues of computational efficiency, and present a ranking abstraction for proving termination.

Two safety properties of interest are preservation and sortedness, expressed as follows:
\[
\begin{gather*}
\forall t .\left(\pi=0 \wedge t \neq 0 \wedge N x t^{*}(x, t)\right) \rightarrow \square\left(N x t^{*}(x, t)\right)  \tag{17}\\
\forall t, s \cdot\left(\pi=11 \wedge N x t^{*}(x, t) \wedge N x t^{*}(t, s)\right) \Rightarrow \operatorname{data}[t] \leq \operatorname{data}[s] \tag{18}
\end{gather*}
\]

As in previous examples we augment the program with a skolem constant for each universal variable. The initial abstraction consists of predicates collected from atomic formulas in properties (17) and (18) and from conditions in the program. These predicates are
\[
\begin{gathered}
\text { last }=N x t[x], y n=\text { last }, \text { data }[y]>\operatorname{data}[y n], \text { prev }=0, t=0, \\
N x t^{*}(x, 0), N x t^{*}(x, t), N x t^{*}(t, s), \operatorname{data}[t] \leq \operatorname{data}[s]
\end{gathered}
\]

This abstraction is too coarse for either property, requiring several iterations of refinement. Since we presently have no heuristic for refinement, new predicates must be derived manually from concretized counterexamples. In shape analysis typical candidates for refinement are reachability properties among program variables that are not expressible in the current abstraction. For example, the initial abstraction cannot express any nontrivial relation among the variables \(x\), last, \(y\), \(y n\), and prev. Indeed, our final abstraction includes, among others, the predicates \(N x t^{*}(x, p r e v)\) and \(N x t^{*}(y n\), last \()\). In the case of prev, \(y\), and \(y n\), it is sufficient to use 1 -step reachability, which is more efficiently computed. Hence we have the predicates \(N x t[\) prev \(]=y\) and \(N x t[y]=y n\).

When abstracting Bubble Sort, one difficulty, in terms of time and memory, is in computing the BDD representation of the abstraction mapping. This becomes apparent as the abstraction is refined with new graph reachability predicates. Naturally, computing the abstract program is also a major bottleneck.

One optimization technique used is to compute a series of increasingly more refined (and complex) abstractions \(\alpha_{1}, \ldots, \alpha_{n}\), with \(\alpha_{n}\) being the desired abstraction. For each \(i=1, \ldots, n-1\), we abstract the program using \(\alpha_{i}\) and compute the set of abstract reachable states. Let \(\varphi_{i}\) be the concretization of this set, which represents the strongest invariant expressible by the predicates in \(\alpha_{i}\). We then proceed to compute the abstraction according to \(\alpha_{i+1}\), while using the invariant \(\varphi_{i}\) to limit the state space. This technique has been invaluable in limiting state explosion, almost doubling the size of models we have been able to handle.

Proving termination of Bubble Sort is more challenging than that of ListREVERSAL due to the nested loop. While a deductive framework would require constructing a global ranking function, the current framework requires only to identify individual rankings of each loop. Therefore we examine both loops independently, specifically their exit conditions.

The outer loop condition (last \(\neq N x t[x])\) implies that "nearness" of last to \(x\) is a measure of progress. We conjecture that after initialization, subsequent assignments advance last "backward" toward \(x\). This suggests the ranking \(\delta_{3}\) defined in Equation (16). As for the inner loop, it iterates while \(y n \neq\) last. We conjecture that \(y n\) generally progresses "forward" toward the list tail. This suggests the ranking \(\delta_{1}\) from Equation (14).


Figure 13: Program AVL-LEFT-Rotate

We use \(\delta_{1}\) and \(\delta_{3}\) as a ranking augmentation, as well as a version of state abstraction described for verifying the safety property that omits predicates related to skolem constants.
7.2. AVL Tree


Figure 14: AvL-Left-Rotate

Finally, Consider the program Avl-Left-Rotate in Fig. 13. The sequence of figures in Fig. 14 demonstrate the AVL tree initial state in Fig. 14(a). Line (3) of the program performs two heap mutations simultaneously in order to redirect the right link of node old_r and the left link of node \(r\). For clarity, these are shown as two steps in Fig. 14(b) and in Fig. 14(c). Fig. 14(d) shows the heap state upon termination of the program.

A property that one may wish to verify is that every node that is reachable from the original left subtree, remains reachable from the left subtree after the rotation. Formally, the property is expressed by:
\[
\begin{equation*}
\forall x . \operatorname{reach}(\operatorname{left}[r], x) \rightarrow \square \operatorname{reach}(\operatorname{left}[r], x) \tag{19}
\end{equation*}
\]

We used our method and obtained an automatic verification of this safety property. Note that since the program has no loops, termination is trivially guaranteed.

\section*{8. Conclusion}

This work presents a "shapeless" shape analysis - an alternative to shape analysis for reasoning about programs that perform destructive updates to heap structures. The
focus is on "single-parent" heap structures and structures that can be easily mapped into them. As we show, this covers the important family of trees (and forests).

Roughly speaking, we prove a small model theorem that allows, given a heap program and a property to be verified, to derive a predicate-abstracted program and property from the original program instantiated on a heap of the size implied by the small model theorem. The predicated abstracted program is obtained by model checking techniques with little interference from the user - the user need only supply the predicate base, which is often trivial. Then, one can check whether the predicated abstracted program satisfies the property and, if so, conclude that so does the original program. The methodology applies to all properties in the assertional language defined here, which covers a large range of properties encountered in the specifications of heap manipulating programs.

The method described here can be applied to multi-linked heaps with unbounded out-degrees, and to heap structures whose "backbone" (obtained when link directions are ignored) is similar to that of single-parent structures. We are currently exploring verification of such structures as well as using multi-linked heap systems as the basis for further structure simulation (as in, e.g., [31, 20]).

Comparing the power of our methodology to other methodologies of analyzing heap structure is also a topic of further research (e.g., [27] ). Obviously, there are properties that, being based on precise abstraction, our method is too weak to deal with. Yet, it offers several advantages over the alternatives, the obvious ones are its simplicity, its elegant handling of termination properties, and its reliance on existing model checking tools. While we used BDD-techniques to obtain the experimental results, SAT and SMT solvers could have been used.

Acknowledgements. We would like to thank the anonymous referees for insightful comments.

\section*{References}
[1] Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Jiazhao Xu, and Lenore D. Zuck. Parameterized verification with automatically computed inductive assertions. In Proc. \(13^{\text {rd }}\) Intl. Conference on Computer Aided Verification, pages 221-234. LNCS 2102, 2001.
[2] Ittai Balaban. Shape Analysis by Augmentation, Abstraction, and Transformation. PhD thesis, New York University, New York, May 2007.
[3] Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. Shape analysis by predicate abstraction. In Radhia Cousot, editor, Proc. of the \(6^{\text {th }}\) Int. Conference on Verification, Model Checking, and Abstract Interpretation, volume 3385 of Lect. Notes in Comp. Sci., pages 164-180. Springer, 2005.
[4] Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. Modular ranking abstraction. Int. J. Found. Comput. Sci., 18(1):5-44, 2007.
[5] Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. Shape analysis of single-parent heaps. In Proc. of the \(8^{\text {th }}\) Int. Conference on Verification, Model Checking, and Abstract Interpretation, pages 91-105, 2007.
[6] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Relative completeness of abstraction refinement for software model checking. In Tools and Algorithms for Construction and Analysis of Systems, pages 158-172, 2002.
[7] Thomas Ball and Sriram K. Rajamani. Automatically validating temporal safety properties of interfaces. Lecture Notes in Computer Science, 2057:103+, 2001.
[8] Michael Benedikt, Thomas W. Reps, and Shmuel Sagiv. A decidable logic for describing linked data structures. In ESOP, pages 2-19, 1999.
[9] Josh Berdine, Byron Cook, Dino Distefano, and Peter W. O’Hearn. Automatic termination proofs for programs with shape-shifting heaps. In CAV, pages 386400, 2006.
[10] Jesse D. Bingham and Zvonimir Rakamaric. A logic and decision procedure for predicate abstraction of heap-manipulating programs. In VMCAI, pages 207-221, 2006.
[11] Egon Börger, Erich Grädel, and Yuri Gurevich. The Classical Decision Problem. Perspectives of Mathematical Logic. Springer-Verlag, 1997. Second printing (Universitext) 2001.
[12] Ahmed Bouajjani, Marius Bozga, Peter Habermehl, Radu Iosif, Pierre Moro, and Tomás Vojnar. Programs with lists are counter automata. In CAV, pages 517-531, 2006.
[13] Ahmed Bouajjani, Cezara Dragoi, Constantin Enea, and Mihaela Sighireanu. A logic-based framework for reasoning about composite data structures. In Mario Bravetti and Gianluigi Zavattaro, editors, CONCUR, volume 5710 of Lecture Notes in Computer Science, pages 178-195. Springer, 2009.
[14] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Proc. IBM Workshop on Logics of Programs, volume 131 of Lect. Notes in Comp. Sci., pages 52-71. Springer-Verlag, 1981.
[15] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In Computer Aided Verification, pages 154-169, 2000.
[16] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. volume 85 of Lect. Notes in Comp. Sci., pages 169-181. Springer-Verlag, 1980.
[17] Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. In STACS, pages 249-260, 1997.
[18] Sumit Gulwani, Tal Lev-Ami, and Mooly Sagiv. A combination framework for tracking partition sizes. In Zhong Shao and Benjamin C. Pierce, editors, POPL, pages 239-251. ACM, 2009.
[19] Neil Immerman, Alexander M. Rabinovich, Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. The boundary between decidability and undecidability for transitiveclosure logics. In CSL, pages 160-174, 2004.
[20] Neil Immerman, Alexander M. Rabinovich, Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. Verification via structure simulation. In Proc. \(16^{\text {th }}\) Intl. Conference on Computer Aided Verification, Lect. Notes in Comp. Sci., pages 281-294. Springer-Verlag, 2004.
[21] Neil D. Jones and Steven S. Muchnick. Flow analysis and optimization of Lisplike structures. In Steven S. Muchnick and Neil D. Jones, editors, Program Flow Analysis: Theory and Applications, chapter 4, pages 102-131. Prentice-Hall, Englewood Cliffs, NJ, 1981.
[22] Yonit Kesten and Amir Pnueli. Verification by augmented finitary abstraction. Information and Computation, 163(1):203-243, 2000.
[23] Nils Klarlund and Michael I. Schwartzbach. Graph types. In Proc. \(20^{\text {th }}\) ACM Symp. Princ. of Prog. Lang., pages 196-205, New York, NY, USA, 1993. ACM Press.
[24] Roman Manevich, Eran Yahav, Ganesan Ramalingam, and Shmuel Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In Radhia Cousot, editor, Proc. of the \(6^{\text {th }}\) Int. Conference on Verification, Model Checking, and Abstract Interpretation, volume 3385 of Lect. Notes in Comp. Sci., pages 181-198. Springer, 2005.
[25] Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In Programming Language Design and Implementation, 2001.
[26] Amir Pnueli, Yaniv Sa'ar, and Lenore D. Zuck. JTLV : A framework for developing verification algorithms. In Proc. \(22^{\text {nd }}\) Intl. Conference on Computer Aided Verification, 2010. To appear.
[27] Andreas Podelski. private communication, January 2010.
[28] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In LICS, pages 55-74, 2002.
[29] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, Proc. 9th International Conference on Computer Aided Verification (CAV'97), volume 1254, pages 72-83. Springer Verlag, 1997.
[30] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, Proc. 9th INternational Conference on Computer Aided Verification (CAV'97), volume 1254, pages 72-83. Springer Verlag, 1997.
[31] Thomas Wies, Viktor Kuncak, Patrick Lam, Andreas Podelski, and Martin C. Rinard. Field constraint analysis. In Proc. of the \(7^{\text {th }}\) Int. Conference on Verification, Model Checking, and Abstract Interpretation, 2006.
[32] Greta Yorsh, Alexander M. Rabinovich, Mooly Sagiv, Antoine Meyer, and Ahmed Bouajjani. A logic of reachable patterns in linked data-structures. In FoSSaCS, pages 94-110, 2006.

\section*{Appendix A. Proof of Theorem 1}

We prove Theorem 1 using a series of claims. The first establishes that wellformed non-sharing multi-linked states are \(\mathcal{S}\)-mapped into well-formed single parent heap states and vice versa.

Lemma 1. Let \(s_{m} \in \Sigma_{m}\), and let \(s_{s}=\mathcal{S}\left(s_{m}\right)\), then
\[
s_{m} \models \text { no_sharing }_{\wedge} \text { well_formed }_{m} \Longleftrightarrow s_{s} \models \text { well_formed }_{s} \wedge \text { ᄀerror }
\]

\section*{Proof:}

In one direction assume that \(s_{m} \models\) no_sharing \(\wedge\) well_formed \(_{m}\). We need to show that \(s_{s}\) satisfies:
\[
\begin{aligned}
& \neg \text { error } \wedge \operatorname{parent}[0]=0 \wedge \bigwedge_{B \in \mathcal{B}}(\neg B[0]) \wedge \\
& \forall i \neq j .(\text { parent }[i]=\text { parent }[j] \neq 0 \rightarrow c t[i] \neq c t[j])
\end{aligned}
\]
where \(\mathcal{B} \subset \mathcal{V}_{s}\) is the set of bool arrays of \(\mathcal{D}_{s}\). The definition of \(\mathcal{S}\) implies the first three conjuncts. As to the fourth note that in a multi-linked heaps each link leads to a single node, thus for every \(i \in[1 . . k]\), every node has at most one \(\operatorname{lin} k_{i}\)-child. Furthermore, the definition of \(\mathcal{S}\) implies that for every node \(u\) and \(v\), and \(i \in[1 . . k], s_{s}[\) parent \(](u)=v\) and \(s_{s}[c t](u)=i\) iff \(s_{m}\left[l i n k_{i}\right](v)=u\). The claim now follows.

The other direction follows immediately from part (6) of the definition of \(\mathcal{S}\).

The following lemma extends the previous one to M -atomic formulea.
Lemma 2. Let \(s_{m} \in \Sigma_{m}\) be a state that satisfies the no_sharing constraint, and let \(s_{s}=\mathcal{S}\left(s_{m}\right)\). Let \(\varphi_{m}\) be a boolean combination of M-atomic formulae over \(\mathcal{D}_{m}\), and let \(\varphi_{s}\) be its transformation into an assertion over \(\mathcal{D}_{s}\). Then: \(s_{m} \models \varphi_{m} \Longleftrightarrow s_{s} \models \varphi_{s}\)

\section*{Proof:}

The claim follows immediately from Lemma 1 for the case that \(\varphi_{m}\) is an M-atomic non-reach and non-cycle formula. For the other cases, we distinguish between:
\(\varphi_{m}\) is of the form \(\operatorname{reach}\left(x_{1}, x_{2}\right)\). Then, \(\varphi_{s}\) is of the from
\[
\left(x_{1} \neq 0 \wedge x_{2} \neq 0 \wedge \operatorname{parent}^{*}\left(x_{2}, x_{1}\right)\right) \vee\left(x_{2}=0\right)
\]

From the definition of \(\mathcal{S}\) it follows that \(s_{s}\left[x_{1}\right]=s_{m}\left[x_{1}\right]\) and \(s_{s}\left[x_{2}\right]=s_{m}\left[x_{2}\right]\). In one direction, assume that \(s_{m} \models \varphi_{m}\). If \(s_{m}\left[x_{2}\right]=0\), then obviously \(s_{s} \models \varphi_{s}\). Otherwise, assume that \(s_{m}\left[x_{2}\right] \neq 0\). Hence, for some \(n \geq 1\) there exist nodes \(s_{m}\left[x_{1}\right]=u_{1}, \ldots, u_{n}=s_{m}\left[x_{2}\right]\) such that for every \(i=1, \ldots, n\), there exists some \(j_{i} \in[1 . . k]\) such that \(s_{m} \models \operatorname{link}_{j_{i}}\left[u_{i}\right]=u_{i+1}\), and \(s_{m}\left[u_{i}\right] \neq 0\). Since \(\mathcal{D}_{m} \models\) nossharing, it follows that for every \(i=1, \ldots, n-1, s_{s}[\) parent \(]\left(u_{i+1}\right)=\) \(u_{i}\). Thus, \(s_{s} \models\) parent \(^{*}\left(u_{n}, u_{1}\right)\). Thus \(s_{s} \models \varphi_{s}\).

In the other direction, assume that \(s_{s} \models \varphi_{s}\). If \(s_{s}\left[x_{1}\right]=0\), then \(s_{s}\left[x_{2}\right]=0\), and then \(s_{m} \models \varphi_{m}\) trivially follows. Assume therefore that \(s_{s}\left[x_{1}\right] \neq 0\). If \(s_{s}\left[x_{2}\right] \neq 0\), an argument, similar to the one used for this case in the other direction, shows that \(s_{m} \models \varphi_{m}\). If \(s_{s}\left[x_{2}\right]=0\), then let \(u \neq 0\) be such that there is a \(s_{s}[\) parent \(]\)-path from \(u\) to \(s_{s}\left[x_{1}\right]\), and for some \(i \in[1 . . k]\), and for every \(y\) either \(s_{s}[\) parent \(](y) \neq u\) or \(s_{s}[c t](y) \neq i\). Thus, \(s_{m}\left[\operatorname{link} k_{i}\right](u) \neq y\) for every \(y\). It thus follows that \(s_{m}\left[\right.\) link \(\left._{i}\right](u)=0\). Similar arguments to the previous direction show that there is a \(\left(\bigcup_{i=1}^{k} l i n k_{i}\right)\)-path from \(s_{m}\left[x_{1}\right]\) to \(u\). We can therefore conclude that \(s_{m} \models\) \(\operatorname{reach}\left(x_{1}, x_{2}\right)\).
\(\varphi_{m}\) is of the form cycle \((x)\). This case is similar to the previous case.
Since the initial condition of \(\mathcal{D}_{m}\) is not a restricted A-assertion, it needs to be dealt with separately:
Corollary 3. Let \(s_{m} \in \Sigma_{m}\) such that \(s_{m} \models\) no_sharing \(\wedge\) well_formed \(_{m}\). Let \(s_{s}=\) \(\mathcal{S}\left(s_{m}\right)\). Then: \(s_{m} \models \Theta_{m} \Longleftrightarrow s_{s} \models \Theta_{s}\)

\section*{Proof:}

As a consequence of the grammar in Fig. 7, \(\Theta_{m}\) is of the form \(\psi \wedge\) nossharing where \(\psi\) is a boolean combination of M-atomic formulae. Subsection 4.2 defines \(\Theta_{s}\) as \(\psi_{s} \wedge\) well_formed \(_{s}\), where \(\psi_{s}\) is the transformation of \(\psi\) by the rules of Subsection 4.2. From Lemma 2 we have that if \(s_{m} \models\) no sharing, then \(s_{m} \models \psi\) iff \(s_{s} \models \psi_{s}\). From definition of mapping \(\mathcal{S}\) we have \(s_{s} \models \neg\) error, and from Lemma 1 we have \(s_{m} \models\) no_sharing \(\wedge\) well_formed \(_{m}\) iff \(s_{s} \models\) नerror \(\wedge\) well_formed \(_{s}\). Thus \(s_{m} \models \Theta_{m}\) iff \(s_{s} \models \Theta_{s}\).

We now extend Lemma 2 to show that transformation of the transition relation preserves the mapping \(\mathcal{S}\) :

Lemma 4. Let \(s_{m} \in \Sigma_{m}\) and \(s_{s}=\mathcal{S}\left(s_{m}\right)\), such that \(s_{m} \models\) no_sharing. Then for any state \(s_{m}^{\prime} \in \Sigma_{m}, \mathcal{S}\left(s_{m}^{\prime}\right)\) is a \(\rho_{s}\)-successor of \(s_{s}\) if \(s_{m}^{\prime}\) is a \(\rho_{m}\)-successor of \(s_{m}\). Furthermore, if \(s_{m}^{\prime}=\) no sharing, then the reverse direction holds as well.

\section*{Proof:}

Let \(s_{m}^{\prime} \in \Sigma_{m}\) be a state such that \(s_{m}^{\prime} \models\) no_sharing. Since \(\rho_{m}\) is a disjunction of clauses, let \(\varphi\left(\mathcal{V}_{m}\right) \wedge \tau\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right) \wedge \operatorname{preserve}\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right)\) be one such arbitrary clause. Then the transformed clause is given by \(\varphi_{s}\left(\mathcal{V}_{s}\right) \wedge \tau_{s}\left(\mathcal{V}_{s}, \mathcal{V}_{s}^{\prime}\right)\), where \(\varphi_{s}\left(\mathcal{V}_{s}\right)\) is the transformation of \(\varphi\left(\mathcal{V}_{m}\right)\) and \(\tau_{s}\left(\mathcal{V}_{s}, \mathcal{V}_{s}^{\prime}\right)\) is the transformation of \(\tau\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right)\) (recall that the preservation conjunct, present in the original clause, is discarded by the transformation, and that \(\tau_{s}\) encapsulates variable preservation clauses).

From Lemma 2 and Corollary 3 we have \(s_{m} \models \varphi\left(\mathcal{V}_{m}\right)\) iff \(s_{s} \models \varphi_{s}\left(\mathcal{V}_{s}\right)\). Let \(s_{s}^{\prime}=\mathcal{S}\left(s_{m}^{\prime}\right)\). It is left to show that \(\left(s_{m}, s_{m}^{\prime}\right) \models \tau\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right) \wedge \operatorname{preserve}\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right)\) iff \(\left(s_{s}, s_{s}^{\prime}\right) \models \tau_{s}\left(\mathcal{V}_{s}, \mathcal{V}_{s}^{\prime}\right)\). Since \(\tau\) is a conjunction of Assign formulas, we show that for each type of atomic Assign formula \(\psi\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right)\) and its transformation \(\psi_{s}\left(\mathcal{V}_{s}, \mathcal{V}_{s}^{\prime}\right)\), \(\left(s_{m}, s_{m}^{\prime}\right) \models \psi\left(\mathcal{V}_{m}, \mathcal{V}_{m}^{\prime}\right) \Longrightarrow\left(s_{s}, s_{s}^{\prime}\right) \models \psi_{s}\left(\mathcal{V}_{s}, \mathcal{V}_{s}^{\prime}\right)\), and if \(s_{m}^{\prime} \models n o\) sharing then the reverse direction holds as well.
\(\psi\) has the form. \(x_{1}^{\prime}=t_{2}\) where \(t_{2}\) is either an index variable or 0 . In this case the claim holds trivially for both directions.
\(\psi\) has the form. \(B^{\prime}[x]\) or \(\neg B^{\prime}[x]\), where \(B\) is a bool array and \(x\) is an index variable. In the case of \(\neg B^{\prime}[x]\), the claim follows trivially. In the case of \(B^{\prime}[x], \psi_{s}\) is the formula \(\left(x=0 \wedge\right.\) error \(\left.^{\prime}\right) \vee\left(x \neq 0 \wedge B^{\prime}[x]\right)\).
1. \(s_{m}^{\prime} \models\) no_sharing. Then \(s_{m}^{\prime} \models \neg B[0]\), and \(s_{s}^{\prime} \models \neg\) error. If \(\left(s_{m}, s_{m}^{\prime}\right) \models B^{\prime}[x]\), then \(x\) cannot be 0 in \(s_{m}\), nor in \(s_{s}\). From \(\mathcal{S}\) we have \(\left(s_{s}, s_{s}^{\prime}\right) \models x \neq 0 \wedge B^{\prime}[x]\). Otherwise, if \(\left(s_{s}, s_{s}^{\prime}\right) \models \psi_{s}\), then the claim follows from the definition of \(\mathcal{S}\) and the fact that error is FALSE in \(s_{s}^{\prime}\).
2. \(s_{m}^{\prime} \not \vDash\) no sharing. Then \(s_{s}^{\prime} \models\) error. If \(s_{m}[x]=0\), then from the definition of \(\mathcal{S}\) we have \(\left(s_{s}, s_{s}^{\prime}\right) \models x=0 \wedge\) error \(^{\prime}\). Thus \(\left(s_{m}, s_{m}^{\prime}\right) \models \psi \Longrightarrow\left(s_{s}, s_{s}^{\prime}\right) \models \psi_{s}\). Otherwise, \(s_{m}[x]=s_{s}[x] \neq 0\). Since, by definition of \(\mathcal{S}, s_{m}^{\prime}[B]\left(s_{m}[x]\right)=\) \(s_{s}^{\prime}[B]\left(s_{s}[x]\right)\), then \(\left(s_{m}, s_{m}^{\prime}\right) \models x \neq 0 \wedge B^{\prime}[x] \operatorname{iff}\left(s_{s}, s_{s}^{\prime}\right) \models x \neq 0 \wedge B^{\prime}[x]\).
\(\psi\) has the form. \(x_{1}^{\prime}=\operatorname{link}_{j}\left[x_{2}\right]\). We focus on the nontrivial case that \(s_{m}\left[x_{2}\right] \neq 0\) and \(s_{m}^{\prime}\left[x_{1}^{\prime}\right] \neq 0\). First assume that \(x_{2}\) is a leaf, i.e., \(s_{m}\left[\operatorname{lin} k_{j}\right]\left(s_{m}\left[x_{2}\right]\right)=0\). In this case \(s_{m}^{\prime}\left[x_{1}\right]=0\), and by definition of \(\mathcal{S}, s_{s}^{\prime}\left[x_{1}\right]=0\). From the assumption, we have \(s_{m} \models \operatorname{link}_{j}\left[x_{1}\right]=0\). Then by Lemma 2, \(s_{s} \models \forall y\). \(\left(\right.\) parent \(\left.[y] \neq x_{2} \vee c t[y] \neq j\right)\). Otherwise, assume that \(x_{2}\) is not a leaf, i.e., \(s_{m}\left[\operatorname{lin} k_{j}\right]\left(s_{m}\left[x_{2}\right]\right) \neq 0\). Then by definition of \(\mathcal{S}\), there exists a node \(u \neq 0\) such that \(s_{m}^{\prime}\left[x_{1}\right]=u\) and \(s_{m}^{\prime}\left[\operatorname{link}_{j}\right]\left(s_{m}\left[x_{2}\right]\right)=u\). Then by definition of \(\mathcal{S}\), \(s_{s}[\) parent \(](u)=s_{s}\left[x_{2}\right], s_{s}[c t](u)=j\), and \(s_{s}^{\prime}\left[x_{1}\right]=u\). Thus \(\left(s_{s}, s_{s}^{\prime}\right) \models \exists y .\left(\operatorname{parent}[y]=x_{2} \wedge c t[y]=j \wedge x_{1}^{\prime}=y\right)\). In the reverse direction, if \(s_{m}\) and \(s_{m}^{\prime}\) both satisfy the no_sharing constraint, then the claim follows trivially from the definition of \(\mathcal{S}\).
\(\psi\) has the form. \(\operatorname{link} k_{j}^{\prime}\left[x_{1}\right]=x_{2}\). Then \(\psi_{s}\) is the formula
\[
\begin{align*}
& \text { Err } \wedge e^{\prime} \text { error }^{\prime}  \tag{1}\\
& \text { V } \\
& {\left[\begin{array}{c}
\quad \neg E r r \\
\wedge \\
\left(x_{2}=0 \vee\left(x_{2} \neq 0 \wedge \operatorname{parent}^{\prime}\left[x_{2}\right]=x_{1} \wedge c t^{\prime}\left[x_{2}\right]=j\right)\right) \\
\wedge\left(\begin{array}{c}
\forall z \cdot\left(\operatorname{parent}[z] \neq x_{1} \vee c t[z] \neq j\right) \\
\vee \exists z \cdot\left(\operatorname{parent}[z]=x_{1} \wedge c t[z]=j \wedge\right. \\
\\
\\
\\
\left.\left(z=x_{2} \vee \operatorname{parent}^{\prime}[z]=0\right)\right)
\end{array}\right]
\end{array}\right]} \tag{2}
\end{align*}
\]

First assume \(\left(s_{m}, s_{m}^{\prime}\right) \models \psi\). Let \(u_{1}=s_{m}\left[x_{1}\right]\) and \(u_{2}=s_{m}\left[x_{2}\right]\). We consider two cases:
1. Node \(u_{2}\) has multiple parents in \(s_{m}^{\prime}\), one of which must be \(u_{1}\). In this case, we have \(s_{m}^{\prime} \models\) no_sharing. Furthermore, by definition of \(\mathcal{S}\), we have \(s_{s}^{\prime}[\) error \(]=\) True and \(s_{s} \models E r r\). Thus \(\left(s_{s}, s_{s}^{\prime}\right) \models \psi_{s}\).
2. Node \(u_{2}\) has a single parent in \(s_{m}^{\prime}\), which must be \(u_{1}\). In this case it must be the case that \(s_{s} \models \neg E r r\). We now show that \(\left(s_{s}, s_{s}^{\prime}\right)\) satisfies the other two conjuncts in disjunct (2) of \(\psi_{s}\). The conjunct \(\left(x_{2}=0 \vee\left(x_{2} \neq 0 \wedge\right.\right.\) parent \(^{\prime}\left[x_{2}\right]=\) \(\left.x_{1} \wedge c t^{\prime}\left[x_{2}\right]=j\right)\) ) follows from the definition of \(\mathcal{S}\). As for the third conjunct,
consider first the case that \(u_{1}\) has no \(j\)-child in \(s_{m}\). Then by definition of \(\mathcal{S}\), \(s_{s} \models \forall z . \operatorname{parent}[z] \neq x_{1} \vee c t[z] \neq j\). Otherwise, there exists a node \(z\) that is the \(j\)-child of \(u_{1}\) in \(s_{m}\). If \(z\) is not \(u_{2}\), then it is no longer the \(j\)-child of \(u_{1}\) in \(s_{m}^{\prime}\). It follows from the definition of \(\mathcal{S}\) that \(\left(s_{s}, s_{s}^{\prime}\right) \models \psi_{s}\).

It is left to show the reverse direction, under the assumption that \(s_{m}^{\prime} \models\) no_sharing. It follows that \(s_{s}^{\prime}[\) error \(]=\) FALSE. Thus, it must be the case that \(\left(s_{s}, s_{s}^{\prime}\right)\) satisfies disjunct (2) of \(\psi_{s}\). Let \(u_{1}=s_{s}\left[x_{1}\right]\) and \(u_{2}=s_{s}\left[x_{2}\right]\). From the definition of \(\mathcal{S}\) and the conjunct \(\left(x_{2}=0 \vee\left(x_{2} \neq 0 \wedge\right.\right.\) parent \(\left.\left.^{\prime}\left[x_{2}\right]=x_{1} \wedge c t^{\prime}\left[x_{2}\right]=j\right)\right)\) we conclude that if \(u_{2} \neq 0\), then \(u_{2}\) is a \(j\)-child of \(u_{1}\) in \(s_{m}^{\prime}\). If \(u_{2}=0\), then from the third conjunct we conclude that \(u_{1}\) has no child in \(s_{m}^{\prime}\). Therefore, \(\left(s_{m}, s_{m}^{\prime}\right) \models \psi\).

Corollary 5. Let \(\mu: s_{m}^{0}, s_{m}^{1}, \ldots\) be a (finite or infinite) sequence of states that consists only of non-sharing states. Then \(\mu\) is a run of \(\mathcal{D}_{m}\) iff \(\mathcal{S}(\mu): \mathcal{S}\left(s_{m}^{0}\right), \mathcal{S}\left(s_{m}^{1}\right) \ldots\) is a run of \(\mathcal{D}_{s}\) without error states.

\section*{Proof:}

The proof is by induction on the run length. At the base case, from Corollary 3 we have that \(\mathcal{S}\left(s_{m}^{0}\right) \models \Theta_{s}\) iff \(s_{m}^{0} \models \Theta_{m}\). Since \(\Theta_{m}\) is defined to include the conjunct no_sharing, then \(s_{m}^{0}\) satisfies the non-sharing constraint, and by definition of \(\mathcal{S}\) we have \(\mathcal{S}\left(s_{m}^{0}\right) \models\) नerror.

For the inductive step, let \(s_{m}^{0}, \ldots, s_{m}^{n}\) be a run of \(\mathcal{D}_{m}\) that is without sharing, and let \(\mathcal{S}\left(s_{m}^{0}\right), \ldots, \mathcal{S}\left(s_{m}^{n}\right)\) be a run of \(\mathcal{D}_{s}\) that is without error states. By Lemma 4 and the definition of \(\mathcal{S}\), a \(\mathcal{D}_{m}\)-state \(s_{m}^{n+1}\) without sharing is a \(\rho_{m}\)-successor of \(s_{m}^{n}\) iff \(\mathcal{S}\left(s_{m}^{n+1}\right)\) is a \(\rho_{s}\)-successor of \(s_{s}\) such that \(\mathcal{S}\left(s_{m}^{n+1}\right)[\) error \(]=\) FALSE.

\section*{Appendix B. Proof of Theorem 4}

Observation 2. Recall that \(N\) is the set of \([0 . . h]\) values that \(M\) assigns to free terms in \(\mathcal{T}_{\varphi}\). For every \(n_{i}, n_{j} \in N\) and every index array \(Z\), the following hold:
1. If \(M(Z)\left[n_{i}\right]=n_{j}\) then \(\bar{M}(Z)[i]=j\), and if \(\bar{M}(Z)[i]=j\) then \(M \models Z^{*}\left(n_{i}, n_{j}\right)\);
2. \(M \not \vDash Z^{*}\left(n_{i}, n_{\ell}\right)\), for any \(n_{\ell} \in N\), iff \(\bar{M}(Z)[i]=m+1\).
3. \(M \models Z^{*}\left(n_{i}, n_{j}\right)\) iff \(\bar{M} \models Z^{*}(i, j)\);
4. If \(B^{\prime}[u]\) occurs in \(\varphi\) for some \(u \in \mathcal{T}_{\varphi}\) and \(a\) bool array \(B \in \mathcal{V}\), then \(u\), parent \([u]\), and parent' \([u]\) are all in \(\mathcal{T}_{\varphi}\).

\section*{Proof:}
(1), (2), and (4) follow immediately from the construction. As for (3), in one direction assume that \(M \models Z^{*}\left(n_{i}, n_{j}\right)\). Thus, there exists a \(Z\)-chain \(\alpha\) : \(n_{i}=v_{0}, \ldots, v_{k}=n_{j}\) in \(M\). Remove all the non- \(N\) nodes from \(\alpha\), and let \(v_{i_{0}}, \ldots, v_{i_{n}}\) be the remaining nodes. From the definition of \(\bar{M}(Z)\) it follows that for every \(\ell=1, \ldots, n, \bar{M}(Z)\left[\gamma\left(v_{i_{\ell-1}}\right)\right]=\)
\(\gamma\left(v_{i_{\ell}}\right)\). Thus, \(\bar{M} \models Z^{*}\left(\gamma\left(v_{0}\right), \gamma\left(v_{k}\right)\right)=Z^{*}(i, j)\). In the other direction assume that \(\bar{M} \models Z^{*}(i, j)\). Since \(n_{i} \in N, i \neq m+1\). Therefore, there exists a \(Z\) chain \(\alpha: i=u_{0}, u_{1}, \ldots, u_{k}=j\) in \(\bar{M}\) such that for every \(\ell=1, \ldots, k, \bar{M}(Z)\left(u_{\ell-1}\right)=u_{\ell}\). From part(1) it now follows that \(M \models Z^{*}\left(n_{i}, n_{j}\right)\).

We now return to the proof of Theorem 4.

\section*{Proof:}

Assume that \(\varphi\) is satisfiable. Recall that \(\varphi\) is a restricted EA-assertion, i.e., \(\varphi\) is of the form \(\varphi: \exists \vec{x} . \psi(\vec{u}, \vec{x})\), where \(\vec{x}\) and \(\vec{u}\) are disjoint lists of index variables, and \(\psi\) is a boolean combination of atomic formulae and restricted A-assertions. A model satisfies \(\varphi\) if it can be augmented by an interpretation of \(\vec{x}\) such that the augmented model satisfies \(\psi(\vec{u}, \vec{x})\). Let \(M\) be such an augmented model, and let \(\bar{M}\) be its reduction with respect to \(\mathcal{T}_{\psi}=\mathcal{T}_{\varphi}\). To prove the theorem, we need to show that \(\bar{M} \models \psi\) if \(M \models \psi\).

Assume therefore that \(M \models \psi\). To show that \(\bar{M} \models \psi\), it suffices to show that (1) every atomic formula \(p\) is true in \(\bar{M}\) iff it is true in \(M\), and (2) every restricted A-assertion \(p\) that is satisfied in \(M\) is also satisfied in \(\bar{M}\). (Recall that restricted Aassertions may only appear in \(\psi\) under positive polarity.)

For the first case, let \(p\) be an atomic sub-formula of \(\psi\). We distinguish between the following cases:
\(p\) is a tcf formula. The claim follows immediately from Observation 2 (part 3).
\(p\) is of the form \(i\)-subtree \({ }_{Z}\left(x_{1}, x_{2}\right) . Z, x_{1}\), and \(x_{2}\) are assumed to be an index array and index variables, respectively. In this case, we are dealing with an ordered heap as defined in Subsection 3.2, and assume the presence of an array \(c t:\) index \(\rightarrow[1 . . k]\), with \(i \in[1 . . k]\). In one direction, assume that \(M \models p\). Expanding the definition of \(p\) to \(\exists u \cdot Z[u]=x_{2} \wedge c t[u]=i \wedge Z^{*}\left(x_{1}, u\right)\), we conclude that \(M \models Z^{*}\left(x_{1}, x_{2}\right)\).
We first identify the \(Z\)-chain from \(x_{1}\) to \(x_{2}\) in \(M\), i.e. the node sequence \(M\left[x_{1}\right]=\) \(u_{1}, \ldots, u_{\ell}, u_{\ell+1}=M\left[x_{2}\right]\) such that \(M[Z]\left(u_{j}\right)=u_{j+1}\), for every \(j=1, \ldots, \ell\). Let \(n_{j}\) be the node \(u_{a}\), for the maximal \(a \in[1 . . \ell]\), such that \(n_{j} \in N\). Then \(u_{\ell}\) is the \(M\) representative of \(Z\) for \(n_{j}\). Since \(M[Z]\left(u_{\ell}\right)=u_{\ell+1}=M\left[x_{2}\right]\), it must be the case that \(M[c t]\left(u_{\ell}\right)=i\). By construction, \(\bar{M}[c t](j)=M[c t]\left(u_{\ell}\right)=i\), and \(\bar{M}[Z](j)=\gamma\left(M[Z]\left(u_{\ell}\right)\right)=\gamma\left(M\left[x_{2}\right]\right)\). Furthermore, from Observation 2 (part 3) we conclude that node \(j\) is \(Z\)-reachable from node \(\bar{M}\left[x_{1}\right]\) in \(\bar{M}\). Thus, \(x_{1}\) is in the \(i^{t h}\) subtree of \(x_{2}\) in \(\bar{M}\), i.e., \(\bar{M} \models \exists u . Z[u]=x_{2} \wedge c t[u]=i \wedge Z^{*}\left(x_{1}, u\right)\), and the claim holds.
In the other direction, assume that \(\bar{M} \models p\). Let \(\bar{M}\left[x_{1}\right]=j<m+1\) and \(\bar{M}\left[x_{2}\right]=\) \(\ell<m+1\). The claim is proven by considering the \(Z\)-chain in \(\bar{M}\) from \(j\) to \(\ell\) and, based on the definition of \(\bar{M}\), constructing a corresponding \(Z\)-chain in \(M\) from \(M\left[x_{1}\right]=n_{j}\) to \(M\left[x_{2}\right]=n_{\ell}\) in which \(n_{j}\) is in the \(i^{t h}\) subtree of \(n_{\ell}\).
\(p\) is a bool variable. The claim follows trivially from the construction of \(\bar{M}\).
\(p\) is of the form \(B[u]\) for an index variable \(u\) and a bool array \(B\). It then follows that parent \([u]\) or parent \({ }^{\prime}[u]\) is in \(\mathcal{T}_{\varphi}\), according to whether \(B\) is unprimed or primed, and then it follows from the construction that \(\bar{M}[B](u)=M[B](u)\).
\(p\) is of the form \(t_{1}=t_{2}\) for index terms \(t_{1}\) and \(t_{2}\). Since \(t_{1}, t_{2} \in \mathcal{T}_{\varphi}\), it follows from the construction that \(M \models t_{1}=t_{2}\) iff \(\bar{M} \models t_{1}=t_{2}\).

For the second case, let \(p\) be a universal formula. We distinguish between two cases. The first is when \(p\) is in one of the forms: \(\forall y . Z[y] \neq u, \forall y . Z[y] \neq u \vee B[y]\), or \(\forall y . Z[y] \neq u \vee \neg B[y]\). We show here the second case; The other two are similar. Recall that \(u\) must be in \(\mathcal{T}_{\varphi}\), and assume that \(M(u)=n_{j}\). Assume, by way of contradiction, that \(M \models \forall y . Z[y] \neq u \vee B[y]\) and for some \(i \in[0 . . m+1], \bar{M} \models Z[i]=j \wedge \neg B[i]\). If \(i=m+1\), then obviously \(M(Z)[i]=m+1\), and thus \(\bar{M} \notin Z[i]=u\). Hence, \(i \neq m+1\). From Observation 2 it follows that \(M[Z]\left(n_{i}\right) \neq n_{j}\). Thus, there exists a \(Z\)-representative \(v \neq n_{i}\) for \(i\) in \(M\). From the construction it follows that \(\bar{M}(Z)[i]=\) \(\gamma(M(Z)[v])\) and that \(\bar{M}(B)[i]=M(B)[v]\). From the assumption that \(\bar{M} \models \neg B[i]\), it follows that \(\neg M(B)[v]\), and from the assumption that \(\bar{M} \models p\) it then follows that \(M(Z)[v] \neq n_{j}\), contradicting the assumption that \(\bar{M}(Z)[i]=j\).

It remains to show the claim for the case that \(p\) is a preservation formula. We distinguish between the following cases:
\(p\) is a preservation formula of an index array. Hence, \(p\) is of the form \(\forall y \cdot Z^{\prime}[y]=\) \(Z[y] \vee \bigvee_{i=j}^{n}\left(y=y_{i}\right)\), where \(y_{1}, \ldots, y_{n}\) are index variables in \(\mathcal{T}_{\varphi}\) and \(Z\) is an index array. Denote by \(Y\) the set \(\left\{y_{1}, \ldots, y_{n}\right\}\) and by \(\gamma(Y)\) the set \(\left\{\gamma\left(y_{1}\right), \ldots, \gamma\left(y_{k}\right)\right\}\). Thus \(p\) can be rewritten as \(\forall y . Z^{\prime}[y]=Z[y] \vee y \in Y\). Assume that \(M \models p\). We have to show that \(\bar{M} \models p\), i.e., that \(\bar{M} \models \forall i \in[0 . . m+1] . Z^{\prime}[i]=Z[i] \vee\) \(\underline{i} \in \gamma(Y)\). Assume, by way of contradiction, that for some \(i \in[0 . . m+1]\), \(\bar{M} \models Z^{\prime}[i] \neq Z[i] \wedge i \notin \gamma(Y)\). We show that \(\bar{M}(Z)[i]=\bar{M}\left(Z^{\prime}\right)[i]\), contradicting the assumption. Since \(\bar{M}(Z)[m+1]=\bar{M}\left(Z^{\prime}\right)[m+1]=m+1\), it follows that \(i \neq m+1\). Consider the \(Z\)-chain \(n_{i}=u_{0}, u_{1}, \ldots\) and the \(Z^{\prime}\)-chain \(n_{i}=v_{0}, v_{1}, \ldots\) in \(M\). Since \(i \notin \gamma(Y)\), it follows, from the assumption that \(M \models p\), that \(M \models Z\left[u_{0}\right]=Z\left[v_{0}\right]\), hence \(v_{1}=u_{1}\). Proceeding like this, we obtain that either
1. For all \(j \geq 0, u_{j}=v_{j}\), or
2. For some \(m \geq 1, u_{m}=v_{m} \in Y\), and for all \(j=0, \ldots, m-1, u_{j}=v_{j} \notin Y\).

In the first case we obtain that \(\bar{M}\left(Z^{\prime}\right)[i]=\bar{M}(Z)[i]\). In the second case, since \(u_{m}=v_{m} \in Y \subseteq \mathcal{T}_{\varphi}\), we obtain that \(i\) has the same \(Z\) - and \(Z^{\prime}\)-representative in \(M\), and thus \(\bar{M}(Z)[i]=\bar{M}\left(Z^{\prime}\right)[i]\). (Note that this \(Z\)-representative is either \(u_{j}=v_{j}\) for some \(j<m\), or \(u_{m}=j_{m}\). The claim follows in either case.)
\(p\) is a preservation formula of a bool array. Following the notation of the previous part, assume \(p\) is of the form \(\forall y \cdot B^{\prime}[y]=B[y] \vee y \in \underline{Y}\) where \(Y\) is a set of index variables in \(\mathcal{T}_{\varphi}\). Assume that \(M \models p\), and that \(\bar{M} \not \vDash p\), i.e., for some \(i \in[0 . . m+1], \bar{M} \models B^{\prime}[i] \neq B[i] \wedge i \notin \gamma(Y)\). Since \(\bar{M}(B)[m+1]=M(B)[d]\), \(\bar{M}\left(B^{\prime}\right)[d]=M\left(B^{\prime}\right)[d]\), and \(d \notin Y\), it follows that \(i \neq m+1\).
This case is handled similarly to the previous case, considering the \(Z\)-chain \(n_{i}=\) \(u_{0}, \ldots\) and \(Z^{\prime}\)-chain \(n_{i}=v_{0}, \ldots\) in \(M\), and concluding that \(\bar{M}\left(B^{\prime}\right)[i]=\bar{M}(B)[i]\). The only difference is in the inductive step: Let \(k \geq 0\), and assume that for all \(j \leq k, u_{j}=v_{j}\) and \(u_{j} \notin Y\). If \(M\left(Z^{\prime}\right)\left[v_{k}\right]=M(Z)\left[v_{k}\right]\), then obviously \(v_{k+1}=u_{k+1}\). Otherwise, \(M\left(Z^{\prime}\right)\left[v_{k}\right] \neq M(Z)\left[v_{k}\right]\). From Observation 2, part
(4), it follows that \(v_{k}, u_{k+1}, v_{k+1} \in \mathcal{T}_{\varphi}\). It thus follows that \(n_{i}\) has the same \(Z\) and \(Z^{\prime}\) representative in \(M\) (which is either \(v_{0}\), \(v_{j}\) for some \(j<k\), or \(v_{k}\) ) and therefore \(\bar{M}(B)[i]=\bar{M}\left(B^{\prime}\right)[i]\).

\section*{Appendix C. Proof of Theorem 7}

Theorem 7 claims that \(M \models \rho_{\delta_{i}}\) implies \(\bar{M} \models \rho_{\delta_{i}}\), where \(\rho_{\delta_{i}}\) is defined as \(\operatorname{dec}^{\prime}=\) decrease \(_{\delta_{i}}\). We prove the claim for a ranking \(\delta_{1}\) of the form \(\delta_{1}(x)=\left\{i \mid N x t^{*}(x, i)\right\}\) specified in Equation (14). The cases of \(\delta_{2}\) and \(\delta_{3}\) are justified by similar arguments.

\section*{Proof:}

The evaluation of \(\delta_{1}\) in \(M\), written \(M\left[\delta_{1}\right]\), is the set \(\left\{i \mid M\left[N x t^{*}\right](M[x], i)\right\}\), i.e, the set of all \(M\)-nodes which are reachable from \(M[x]\) by \(M[N x t]\)-links. The evaluation of \(\delta_{1}\) in \(\bar{M}\) and of \(\delta_{1}^{\prime}\) in \(M\) and \(\bar{M}\) are defined similarly.

First note the following property of terms in \(\mathcal{T}\) : It follows directly from Property P5 of Theorem 4 that, for any term \(t\) in \(\mathcal{T}\) and \(\delta \in\left\{\delta_{1}, \delta_{1}^{\prime}\right\}, M[t] \in M[\delta]\) iff \(\bar{M}[t] \in \bar{M}[\delta]\).

To prove the claim it is enough to show that both properties \(\delta_{1} \supset \delta_{1}^{\prime}\) and \(\delta_{1}=\delta_{1}^{\prime}\) are satisfied by \(M\) iff they are satisfied by \(\bar{M}\). First assume \(M \models \delta_{1} \supset \delta_{1}^{\prime}\). It is easy to show that \(\delta_{1} \supseteq \delta_{1}^{\prime}\) is satisfied in \(\bar{M}\). This is true since by construction, any node \(i \in[0 \ldots N]\) is pointed to in \(\bar{M}\) by a term in \(\mathcal{T}\), and membership in \(\delta_{1}, \delta_{1}^{\prime}\) is preserved for such terms.

It is left to show that \(\delta_{1} \neq \delta_{1}^{\prime}\) is satisfied in \(\bar{M}\). We do this by identifying a term in \(\mathcal{T}\) that \(M\) interprets as a node in \(M\left[\delta_{1}\right]-M\left[\delta_{1}^{\prime}\right]\). Such a term must point to a node in \(\bar{M}\) that is a member of \(\bar{M}\left[\delta_{1}\right]-\bar{M}\left[\delta_{1}^{\prime}\right]\). To perform this identification, let \(\ell\) be a node in \(M\left[\delta_{1}\right]-M\left[\delta_{1}^{\prime}\right]\). Let \(M[x]=r_{1}, \ldots, r_{q}=\ell\) denote the shortest \(N x t\)-path in \(M\) from the node \(M[x]\) to \(\ell\), i.e., for \(i=1, \ldots, q-1, M[N x t]\left(r_{i}\right)=r_{i+1}\). Let \(j\) be the maximal index in \([1 . . q]\) such that \(r_{j} \in\left\{n_{0}, \ldots, n_{m}\right\}\), i.e., \(r_{j}\) is the \(M\)-image of some term \(t \in \mathcal{T}\). If \(r_{j} \notin M\left[\delta_{1}^{\prime}\right]\), our identification is complete.

Assume therefore that \(r_{j} \in M\left[\delta_{1}^{\prime}\right]\). According to our construction, there exists an \(M[N x t]\)-chain connecting \(r_{j}\) to \(\ell\), proceeding along \(r_{j+1}, r_{j+2}, \ldots, \ell\). Consider the chain of \(M\left[N x t^{\prime}\right]\)-links starting from \(r_{j}\). At one of the intermediate nodes: \(r_{j}, \ldots, \ell\), the \(M[N x t]\)-chain and the \(M\left[N x t^{\prime}\right]\)-chain must diverge, otherwise \(\ell\) would also belong to \(M\left[\delta_{1}^{\prime}\right]\). Assume that the two chains diverge at \(r_{k}\), for some \(j \leq k<q\). Then, according to strong uniformity (implied by history closure), \(r_{k+1} \in\left\{n_{0}, \ldots, n_{m}\right\}\), contradicting the assumed maximality of \(j\).

In the other direction, assume that \(\bar{M}\) satisfies \(\delta_{1} \supset \delta_{1}^{\prime}\). We first show that \(M\) satisfies \(\delta_{1} \supseteq \delta_{1}^{\prime}\). Let \(n\) be a node in \(M\left[\delta_{1}^{\prime}\right]\), and consider a \(N x t^{\prime}\)-path from \(M\left[x^{\prime}\right]\) to \(n\) in \(M\). Let \(m\) be the ancestor nearest to \(n\) that is pointed to by a term in \(\mathcal{T}\). From Theorem 4 it follows that \(m \in M\left[\delta_{1}\right]\). The fact \(n \in M\left[\delta_{1}\right]\) follows by induction on path length from \(m\) to \(n\) and by uniformity of \(M\) and \(\bar{M}\). Therefore \(M\left[\delta_{1}\right] \supseteq M\left[\delta_{1}^{\prime}\right]\). We now show that \(M\) satisfies \(\delta_{1} \supset \delta_{1}^{\prime}\). Let \(j\) be a node such that \(j \in \bar{M}\left[\delta_{1}\right]-\bar{M}\left[\delta_{1}^{\prime}\right]\). By construction, \(j\) is pointed to in \(\bar{M}\) by a term \(t\) or \(j=m+1\). In the first case, \(t\) points
to a node \(n_{j}\) in \(M\), such that \(n_{j} \in M\left[\delta_{1}\right]-M\left[\delta_{1}^{\prime}\right]\), and we are done. In the latter case, from construction we have \(\bar{M}[N x t](m+1)=\bar{M}\left[N x t^{\prime}\right](m+1)=m+1\). Therefore, if \(m+1\) is not \(N x t^{\prime}\)-reachable from \(\bar{M}\left[x^{\prime}\right]\), there must exist a node \(i\) in \(\bar{M}\left[\delta_{1}\right]-\bar{M}\left[\delta_{1}^{\prime}\right]\) such that \(\bar{M}[N x t](i) \neq \bar{M}\left[N x t^{\prime}\right](i)\). By uniformity, \(i\) must be pointed to in \(\bar{M}\) by a term in \(\mathcal{T}\). From Theorem 4 there exists a corresponding node in \(M\).

It is left to show that \(M \models\left(\delta_{1}=\delta_{1}^{\prime}\right)\) iff \(\bar{M} \models\left(\delta_{1}=\delta_{1}^{\prime}\right)\). This is done by similar arguments.

The case of \(\delta_{2}\), while not presented here, is shown by generalization: While \(\delta_{1}\) involves nodes reachable from a single distinguished pointer \(x, \delta_{2}\) involves nodes on a path between \(x\) and a pointer \(y\). Thus, given node \(\ell\) satisfying some combination of properties of membership in \(\delta_{2}, \delta_{2}^{\prime}\), we identify a node satisfying the same properties, that is also pointed to by a term in \(\mathcal{T}\). Here, however, we consider not only distant ancestors of \(\ell\) on the path from \(x\), but also distant successors on the path to \(y\).```

