arXiv:1301.4372v1 [cs.LO] 18 Jan 2013

A Decidable Theory of Skiplists of Unbounded Size and Arbitrary Height

César Sánchez[1,2] and Alejandro Sánchez[1]

[1] IMDEA Software Institute, Madrid, Spain
[2] Institute for Information Security, CSIC, Spain
{cesar.sanchez,alejandro.sanchez}@imdea.org

Abstract. This paper presents a theory of skiplists of arbitrary height, and shows decidability of the satisfiability problem for quantifier-free formulas.
A skiplist is an imperative software data structure that implements sets by maintaining several levels of ordered singly-linked lists in memory, where each level is a sublist of its lower levels. Skiplists are widely used in practice because they offer a performance comparable to balanced binary trees, and can be implemented more efficiently. To achieve this performance, most implementations dynamically increment the height (the number of levels). Skiplists are difficult to reason about because of the dynamic size (number of nodes) and the sharing between the different layers. Furthermore, reasoning about dynamic height adds the challenge of dealing with arbitrary many levels.
The first contribution of this paper is the theory TSL that allows to express the heap memory layout of a skiplist of arbitrary height. The second contribution is a decision procedure for the satisfiability problem of quantifier-free TSL formulas. The last contribution is to illustrate the formal verification of a practical skiplist implementation using this decision procedure.

1 Introduction

A skiplist [8] is a data structure that implements sets, maintaining several sorted singly-linked lists in memory. Skiplists are structured in levels, where each level consists of a singly-linked list. Each node in a skiplist stores a value and at least the pointer corresponding to the list at the lowest level. Some nodes also contain pointers at higher levels, pointing to the next node present at that level. The "skiplist property" establishes that the lowest level (backbone) list is ordered, and that list at level $i+1$ is a sublist of the list at level i. Search in skiplists is (probabilistically) logarithmic. The advantage of skiplists compared to balanced search trees is that skiplists are simpler and more efficient to implement.

Consider the skiplist layout in Fig. 1. Higher-level pointers allow to *skip* many elements of the backbone list during the search. A search is performed from left to right in a top down fashion, progressing as much as possible in a level before descending. Fig. 1 shows in red the nodes traversed when looking value 88. The

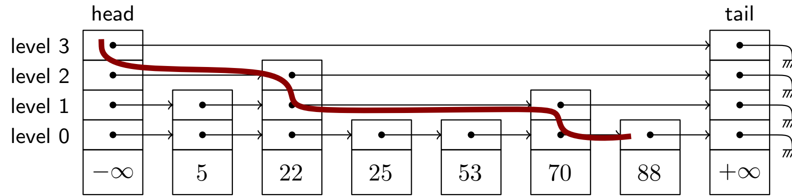

Fig. 1. A skiplist with 4 levels, and the traversal searching 88 (in red).

search starts at level 3 of node *head*, that points to node *tail*, which stores value $+\infty$, greater than 88. Consequently, the search continues at *head* by moving down one level to level 2. The successor of *head* at level 2 stores value 22, which is smaller than 88. Hence, the search continues at level 2 from the node storing 22 until a value greater than 88 is found. The expected logarithmic search of skiplists follows from the probability of a node being present at a certain level decreasing by 1/2 as the level increases (see [8] for an analysis of the running time of skiplists).

In practice, implementations of skiplists vary the height dynamically maintaining a variable that stores the current highest level of any node in the skiplist. The theory TSL presented in this paper allows to automatically proof verification conditions of skiplists with height unbounded (as indicated by a this variable).

We are interested in the formal verification of implementations of skiplists, which requires to reason about unbounded mutable data stored in the heap. One popular approach to the verification of heap programs is Separation Logic [10]. Skiplists, however, are problematic for separation-like approaches due to the aliasing and memory sharing between nodes at different levels. Most of the work in formal verification of pointer programs follows program logics in the Hoare tradition, either using separation logic or with specialized program logics to deal with the heap and pointer structures [1,4,5,13]. Our approach is complementary, consisting on the design of specialized decision procedures for memory layouts which can be incorporated into a reasoning system for proving temporal properties, in the style of Manna-Pnueli [6]. In particular for proving liveness properties we advocate the use of general verification diagrams [2],which allow a clean separation between the temporal reasoning with the reasoning about the data being manipulated. Proofs (of both safety and liveness properties) are ultimately decomposed into verification conditions (VCs) in the underlying theory of state assertions. This paper studies the automatic verification of VCs involving the manipulation of skiplist memory layouts. For illustration purposes we restrict the presentation in this paper to safety properties.

Logics like [1,4,13] are very powerful to describe pointer structures, but they require the use of quantifiers to reach their expressive power. Hence, these logics preclude their combination with methods like Nelson-Oppen [7] or BAPA [3] with other aspects of the program state. Instead, our solution use specific theories of memory layouts [9, 11, 12] that allow to express powerful properties in the quantifier-free fragment using built-in predicates.

For example, in [12] we presented TSL_K, a family of theories of skiplists of fixed height, which are unrolled into the theory of ordered singly-linked lists [11]. Limiting the height of the skiplist (for example to a maximum of 32 levels) would enable to use of TSL_K for verification of such implementations but unfortunately, the model search involved in the automatic proofs of TSL_K VCs is only practical for much lower heights. Handling dynamic height was still an open problem that precluded the verification of practical skiplist implementations. We solve this open problem here with TSL. The theory TSL we present in this paper allows us to reduce the verification of a skiplist of arbitrary height to verification conditions of TSL_K, where the value of K is small and *independent of the skiplist height* in any state of any implementation.

The rest of the paper is structured as follows. Section 2 presents a running example of a program that manipulates skiplists. Section 3 introduces TSL: the theory of skiplists of arbitrary height. Section 4 includes the decidability proof. Section 5 provides some examples of the use of TSL in the verification of skiplists. Finally, Section 6 concludes the paper. Some proofs are missing due to space limitation and are included in the appendix.

2 A Skiplist Implementation

Fig. 2 shows the pseudo-code of a sequential implementation of a skiplist, whose basic classes are *Node* and *SkipList*. Each node stores a key (in the field *key*) for keeping the list ordered, a field *val* containing the actual value stored, and a field *next*: an array of arbitrary length containing the addresses of the following nodes at each level. An entry in *next* at index i points to the successor node at level i. Given an object *sl* of class *SkipList*, we use *sl.head*, *sl.tail* and *sl.maxLevel* for the data members storing the head node, the tail node and the maximum level in use (resp.) When the *SkipList* object *sl* is clear from the context, we use *head*, *tail* and *maxLevel* instead of *sl.head*, *sl.tail* and *sl.maxLevel*. The program in Fig.2 allows executions in which the height of a skiplist, as stored in *maxLevel*, can grow beyond any bound. Finally, nodes contain a ghost field *level* storing the highest level of *next*. We use the @ symbol to denote a ghost field and boxes to describe ghost code. This extra "ghost" code is only added for verification purposes and does not influence the execution of the existing program (it does not affect the control flow or non-ghost data), and it is removed during compilation. Objects of *SkipList* maintain one ghost field *reg* to represent the region of the heap (set of addresses) managed by the skiplist. In this implementation, *head* and *tail* are sentinel nodes for the first and last nodes of the list, initialized with $key = -\infty$ and $key = +\infty$ (resp.) These nodes are not removed during the execution and their *key* field remains unchanged. The amount of ghost code introduced for verification is very small, containing only the book-keeping of the region *reg*.

Fig. 2 shows the algorithms for insertion (Insert), search (Search) and removal (Remove). Fig. 2 also shows the most general client MGC, a program that non-deterministically performs calls to skiplist operations. In this imple-

```
1:  procedure MGC(SkipList sl)
2:      while true do
3:          v := NondetPickValue
4:          nondet
                ⎡ call Insert(sl, v)  ⎤
                ⎢     or              ⎥
5:              ⎢ call Search(sl, v)  ⎥
                ⎢     or              ⎥
                ⎣ call Remove(sl, v)  ⎦
6:      end while
7:  end procedure

8:  procedure Insert(SkipList sl, Value v)
9:      Array⟨Node*⟩ upd
10:     Int lvl := randomLevel
11:     Bool valueWasIn := false
12:     if lvl > sl.maxLevel then
13:         for i := (sl.maxLevel + 1) to lvl do
14:             sl.head.next[i] := sl.tail
15:             sl.tail.next[i] := null
16:         end for
17:         sl.maxLevel := lvl
18:     end if
19:     Node* pred := sl.head
20:     Node* curr := pred.next[sl.maxLevel]
21:     Int i := sl.maxLevel
22:     while 0 ≤ i ∧ ¬valueWasIn do
23:         curr := pred.next[i]
24:         while curr.val < v do
25:             pred := curr
26:             curr := pred.next[i]
27:         end while
28:         upd[i] := pred
29:         i := i − 1
30:         valueWasIn := (curr.val = v)
31:     end while
32:     if ¬valueWasIn then
33:         x := CreateNode(lvl, v)
34:         for i := 0 to lvl do
35:             x.next[i] := upd[i].next[i]
36:             upd[i].next[i] := x
                ┌──────────────────────────┐
                │ if i = 0 then            │
                │     sl.reg := sl.reg ∪ {x} │
                └──────────────────────────┘
37:         end for
38:     end if
39:     return ¬valueWasIn
40: end procedure

41: procedure Search(SkipList sl, Value v)
42:     Int i := sl.maxLevel
43:     Node* pred := sl.head
44:     Node* curr := pred.next[i]
45:     while 0 ≤ i ∧ curr.val ≠ v do
46:         curr := pred.next[i]
47:         while curr.val < v do
48:             pred := curr
49:             curr := pred.next[i]
50:         end while
51:         i := i − 1
52:     end while
53:     return curr.val = v
54: end procedure

55: procedure Remove(SkipList sl, Value v)
56:     Array⟨Node*⟩[sl.maxLevel + 1] upd
57:     Int removeFrom := sl.maxLevel
58:     Node* pred := sl.head
59:     Node* curr := pred.next[sl.maxLevel]
60:     for i := sl.maxLevel downto 0 do
61:         curr := pred.next[i]
62:         while curr.val < v do
63:             pred := curr
64:             curr := pred.next[i]
65:         end while
66:         if curr.val ≠ v then
67:             removeFrom := i − 1
68:         end if
69:         upd[i] := pred
70:     end for
71:     Bool valueWasIn := (curr.val = v)
72:     if valueWasIn then
73:         for i := removeFrom downto 0 do
74:             upd[i].next[i] := curr.next[i]
                ┌─────────────────────────────┐
                │ if i = 0 then               │
                │     sl.reg := sl.reg \ {curr} │
                └─────────────────────────────┘
75:         end for
76:         free (curr)
77:     end if
78:     return valueWasIn
79: end procedure
```

```
class Node     { Value val; Key key; Array⟨Node*⟩ next; Int @level;     }
class SkipList { Node* head; Node* tail; Int @maxLevel; Set⟨Addr⟩ @reg; }
```

Fig. 2. Most general client, Insert, Search and Remove algorithms for skiplists, and the classes *Node* and *SkipList*.

mentation, we assume that the initial program execution begins with an empty skiplist containing only *head* and *tail* nodes at level 0 has already been created. New nodes are then added using the INSERT operation. Since MGC can execute all possible sequence of calls, it can be used to verify properties like method termination or skiplist-shape preservation. The program updates the ghost field *reg* to represent the set of nodes that forms the skiplist at every state. That is: (a) a new node becomes part of the skiplist as soon as it is connected at level 0 in INSERT (line 36); and (b) a node that is being removed stops being part of the skiplist when it is disconnected at level 0 in REMOVE (line 74). For simplicity, we assume in this paper that the fields *val* and *key* within an object of type *Node* contain the same object. A crucial property that we wish to prove of this implementation is that the memory layout maintained by the algorithm is that of a "skiplist": the lower level is an ordered acyclic single linked list, all levels are subset of lower levels, and all the elements stored are precisely those stored in addresses contained in region *reg*.

3 The Theory of Skiplists of Arbitrary Height: TSL

We present in this section TSL: a theory to reason about skiplists of arbitrary height. Formally, TSL is a combination of different theories.

We begin with a brief overview of notation and concepts. A signature Σ is a triple (S, F, P) where S is a set of sorts, F a set of functions and P a set of predicates. If $\Sigma_1 = (S_1, F_1, P_1)$ and $\Sigma_2 = (S_2, F_2, P_2)$, we define $\Sigma_1 \cup \Sigma_2 = (S_1 \cup S_2, F_1 \cup F_2, P_1 \cup P_2)$. Similarly we say that $\Sigma_1 \subseteq \Sigma_2$ when $S_1 \subseteq S_2$, $F_1 \subseteq F_2$ and $P_1 \subseteq P_2$. If $t(\varphi)$ is a term (resp. formula), then we denote with $V_\sigma(t)$ (resp. $V_\sigma(\varphi)$) the set of variables of sort σ occurring in t (resp. φ). Similarly, we denote with $C_\sigma(t)$ (resp. $C_\sigma(\varphi)$) the set of constants of sort σ occurring in t (resp. φ).

A Σ-interpretation is a map from symbols in Σ to values. A Σ-structure is a Σ-interpretation over an empty set of variables. A Σ-formula over a set X of variables is satisfiable whenever it is true in some Σ-interpretation over X. Let Ω be a signature, $\mathcal{A}$ an Ω-interpretation over a set $V$of variables, $\Sigma \subseteq \Omega$ and $U \subseteq V$. $\mathcal{A}^{\Sigma,U}$ denotes the interpretation obtained from $\mathcal{A}$ restricting it to interpret only the symbols in Σ and the variables in U. We use $\mathcal{A}^{\Sigma}$ to denote $\mathcal{A}^{\Sigma,\emptyset}$. A Σ-theory is a pair $(\Sigma, \mathbf{A})$ where Σ is a signature and $\mathbf{A}$ is a class of Σ-structures. Given a theory $T = (\Sigma, \mathbf{A})$, a T-interpretation is a Σ-interpretation $\mathcal{A}$ such that $\mathcal{A}^{\Sigma} \in \mathbf{A}$. Given a Σ-theory T, a Σ-formula φ over a set of variables X is T-satisfiable whenever it is true on a T-interpretation over X.

Formally, the theory of skiplists of arbitrary height is defined as $\mathsf{TSL} = (\Sigma_{\mathsf{TSL}}, \mathbf{TSL})$, where Σ_{TSL} is the union of the following signatures, shown in Fig. 3

$$\Sigma_{\mathsf{TSL}} = \Sigma_{\mathsf{level}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{array}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{reach}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{bridge}}$$

and **TSL** is the class of Σ_{TSL}-structures satisfying the conditions listed in Fig. 4.

Informally, sort addr represents addresses; elem the universe of elements that can be stored in the skiplist; level the levels of a skiplist; ord the ordered keys used to preserve a strict order in the skiplist; array corresponds to arrays of addresses, indexed by levels; cell models *cells* representing objects of class *Node*; mem models the heap, a map from addresses to cells; path describes finite sequences of non-repeating addresses to model non-cyclic list paths, while set models sets of addresses—also known as regions.

The symbols in Σ_{set} are interpreted according to their standard interpretations over set of addresses. Σ_{level} contains symbols 0 and s to build the natural numbers with the usual order. Σ_{ord} models the order between elements, and contains two special elements $-\infty$ and $+\infty$ for the lowest and highest values in the

Signt	Sort	Functions	Predicates
Σ_{level}	level	0 : level s : level $\to$ level	$<$: level $\times$ level
Σ_{ord}	ord	$-\infty, +\infty$: ord	$\preceq$: ord $\times$ ord
Σ_{array}	array level addr	$_[_]$: array $\times$ level $\to$ addr $_\{_ \leftarrow _\}$: array $\times$ level $\times$ addr $\to$ array	
Σ_{cell}	cell elem ord array addr level	$error$: cell $mkcell$: elem $\times$ ord $\times$ array $\times$ level $\to$ cell $_.data$: cell $\to$ elem $_.key$: cell $\to$ ord $_.arr$: cell $\to$ array $_.max$: cell $\to$ level	
Σ_{mem}	mem addr cell	$null$: addr rd : mem $\times$ addr $\to$ cell upd : mem $\times$ addr $\times$ cell $\to$ mem	
Σ_{reach}	mem addr path	ϵ : path $[_]$: addr $\to$ path	$append$: path $\times$ path $\times$ path $reach$: mem $\times$ addr $\times$ addr $\times$ level $\times$ path
Σ_{set}	addr set	$\emptyset$: set $\{_\}$: addr $\to$ set $\cup, \cap, \setminus$: set $\times$ set $\to$ set	$\in$: addr $\times$ set $\subseteq$: set $\times$ set
Σ_{bridge}	mem addr set path level	$path2set$: path $\to$ set $addr2set$: mem $\times$ addr $\times$ level $\to$ set $getp$: mem $\times$ addr $\times$ addr $\times$ level $\to$ path	$ordList$: mem $\times$ path $skiplist$: mem $\times$ set $\times$ level $\times$ addr $\times$ addr

Fig. 3. The signature of the TSL theory

Each sort σ in Σ_{TSL} is mapped to a non-empty set $\mathcal{A}_\sigma$ such that:

(a) $\mathcal{A}_{\mathsf{addr}}$ and $\mathcal{A}_{\mathsf{elem}}$ are discrete sets
(b) $\mathcal{A}_{\mathsf{level}}$ is the naturals with order
(c) $\mathcal{A}_{\mathsf{ord}}$ is a total ordered set
(d) $\mathcal{A}_{\mathsf{array}} = \mathcal{A}_{\mathsf{addr}}^{\mathcal{A}_{\mathsf{level}}}$
(e) $\mathcal{A}_{\mathsf{cell}} = \mathcal{A}_{\mathsf{elem}} \times \mathcal{A}_{\mathsf{ord}} \times \mathcal{A}_{\mathsf{array}} \times \mathcal{A}_{\mathsf{level}}$
(f) $\mathcal{A}_{\mathsf{path}}$ is the set of all finite sequences of (pairwise) distinct elements of $\mathcal{A}_{\mathsf{addr}}$
(g) $\mathcal{A}_{\mathsf{mem}} = \mathcal{A}_{\mathsf{cell}}^{\mathcal{A}_{\mathsf{addr}}}$
(h) $\mathcal{A}_{\mathsf{set}}$ is the power-set of $\mathcal{A}_{\mathsf{addr}}$

Signature	Interpretation
Σ_{level}	• $0^{\mathcal{A}} = 0$ • $s^{\mathcal{A}}(l) = s(l)$, for each $l \in \mathcal{A}_{\mathsf{level}}$
Σ_{ord}	• $x \preceq^{\mathcal{A}} y \wedge y \preceq^{\mathcal{A}} x \to x = y$ • $x \preceq^{\mathcal{A}} y \vee y \preceq^{\mathcal{A}} x$ • $x \preceq^{\mathcal{A}} y \wedge y \preceq^{\mathcal{A}} z \to x \preceq^{\mathcal{A}} z$ • $-\infty^{\mathcal{A}} \preceq^{\mathcal{A}} x \wedge x \preceq^{\mathcal{A}} +\infty^{\mathcal{A}}$ for any $x, y, z \in \mathcal{A}_{\mathsf{ord}}$
Σ_{array}	• $A[l]^{\mathcal{A}} = A(l)$ • $A\{l \leftarrow a\}^{\mathcal{A}} = A_{new}$, where $A_{new}(l) = a$ and $A_{new}(i) = A(i)$ for $i \neq l$ for each $A, A_{new} \in \mathcal{A}_{\mathsf{array}}$, $l \in \mathcal{A}_{\mathsf{level}}$ and $a \in \mathcal{A}_{\mathsf{addr}}$
Σ_{cell}	• $mkcell^{\mathcal{A}}(e, k, \overrightarrow{a}, l) = \langle e, k, A, l \rangle$ • $error^{\mathcal{A}}.arr^{\mathcal{A}}(l) = null^{\mathcal{A}}$ • $\langle e, k, A, l \rangle.data^{\mathcal{A}} = e$ • $\langle e, k, A, l \rangle.key^{\mathcal{A}} = k$ • $\langle e, k, A, l \rangle.arr^{\mathcal{A}} = A$ • $\langle e, k, A, l \rangle.max^{\mathcal{A}} = l$ for each $e \in \mathcal{A}_{\mathsf{elem}}$, $k \in \mathcal{A}_{\mathsf{ord}}$, $A \in \mathcal{A}_{\mathsf{array}}$, and $l \in \mathcal{A}_{\mathsf{level}}$
Σ_{mem}	• $rd(m, a)^{\mathcal{A}} = m(a)$ • $upd^{\mathcal{A}}(m, a, c) = m_{a \mapsto c}$ • $m^{\mathcal{A}}(null^{\mathcal{A}}) = error^{\mathcal{A}}$ for each $m \in \mathcal{A}_{\mathsf{mem}}$, $a \in \mathcal{A}_{\mathsf{addr}}$ and $c \in \mathcal{A}_{\mathsf{cell}}$
Σ_{reach}	• $\epsilon^{\mathcal{A}}$ is the empty sequence • $[a]^{\mathcal{A}}$ is the sequence containing $a \in \mathcal{A}_{\mathsf{addr}}$ as the only element • $([a_1 \mathinner{..} a_n], [b_1 \mathinner{..} b_m], [a_1 \mathinner{..} a_n, b_1 \mathinner{..} b_m]) \in append^{\mathcal{A}}$ iff $a_k \neq b_l$. • $(m, a_{init}, a_{end}, l, p) \in reach^{\mathcal{A}}$ iff $a_{init} = a_{end}$ and $p = \epsilon$, or there exist addresses $a_1, \ldots, a_n \in \mathcal{A}_{\mathsf{addr}}$ such that: (a) $p = [a_1 \mathinner{..} a_n]$ (c) $m(a_r).arr^{\mathcal{A}}(l) = a_{r+1}$, for $r < n$ (b) $a_1 = a_{init}$ (d) $m(a_n).arr^{\mathcal{A}}(l) = a_{end}$
Σ_{bridge}	for each $m \in \mathcal{A}_{\mathsf{mem}}$, $p \in \mathcal{A}_{\mathsf{path}}$, $l \in \mathcal{A}_{\mathsf{level}}$, $a_i, a_e \in \mathcal{A}_{\mathsf{addr}}$, $r \in \mathcal{A}_{\mathsf{set}}$ • $path2set^{\mathcal{A}}(p) = \{a_1, \ldots, a_n\}$ for $p = [a_1, \ldots, a_n] \in \mathcal{A}_{\mathsf{path}}$ • $addr2set^{\mathcal{A}}(m, a, l) = \left\{a' \mid \exists p \in \mathcal{A}_{\mathsf{path}} \;.\; (m, a, a', l, p) \in reach^{\mathcal{A}}\right\}$ • $getp^{\mathcal{A}}(m, a_i, a_e, l) = p$ if $(m, a_i, a_e, l, p) \in reach^{\mathcal{A}}$, and ϵ otherwise • $ordList^{\mathcal{A}}(m, p)$ iff $p = \epsilon$ or $p = [a]$, or $p = [a_1, \ldots, a_n]$ with $n \geq 2$ and $m(a_j).key^{\mathcal{A}} \preceq m(a_{j+1}).key^{\mathcal{A}}$ for all $1 \leq j < n$, for any $m \in \mathcal{A}_{\mathsf{mem}}$ • $skiplist^{\mathcal{A}}(m, r, l, a_i, a_e)$ iff $\left[\begin{array}{c} ordList^{\mathcal{A}}(m, getp^{\mathcal{A}}(m, a_i, a_e, 0)) \quad \wedge \\ r = addr2set^{\mathcal{A}}(m, a_i, 0) \quad \wedge \\ 0 \leq l \wedge \forall a \in r \;.\; m(a).max^{\mathcal{A}} \leq l \quad \wedge \\ m(a_e).arr^{\mathcal{A}}(l) = null^{\mathcal{A}} \quad \wedge \\ \left(0 = l_{max}\right) \vee \\ \big(\exists l_p \;.\; s^{\mathcal{A}}(l_p) = l \wedge \forall\, i \in 0, \ldots, l_p \;. \\ m(a_e).arr^{\mathcal{A}}(i) = null^{\mathcal{A}} \wedge \\ path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m, a_i, a_e, s^{\mathcal{A}}(i))) \subseteq \\ path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m, a_i, a_e, i))\big) \end{array}\right]$

Fig. 4. Characterization of a TSL-interpretation $\mathcal{A}$

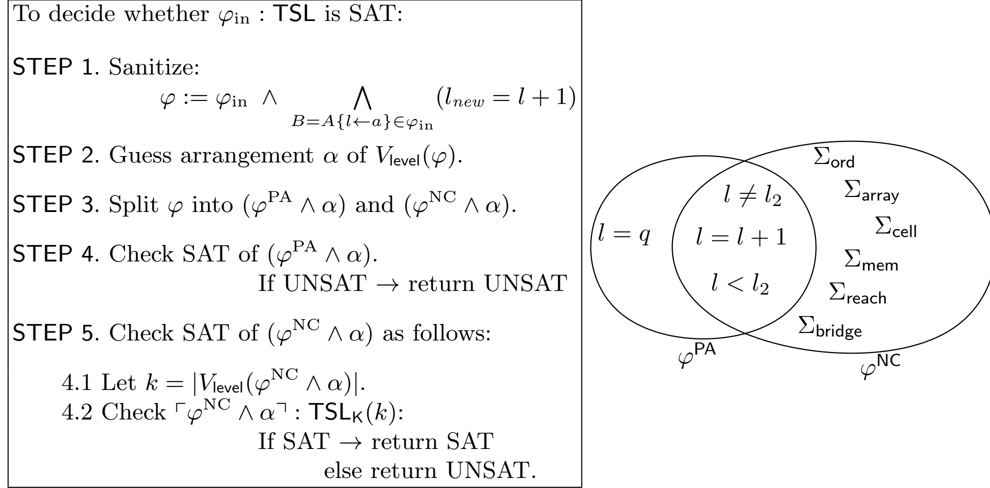

Fig. 5. A decision procedure for the satuasfibility of TSL formulas (left). A split of φ obtained after STEP 1 into φ^{PA} and φ^{NC} (right).

order $\preceq$. Σ_{array} is the theory of arrays defining two operations: $A[i]$ to capture the element of sort addr stored in array A at position given by i of sort level, and $A\{i \leftarrow a\}$ for an array write, which defines the array that results from A by replacing the element at position i with a. Σ_{cell} contains the constructors and selectors for building and inspecting cells, including *error* for incorrect dereferences. Σ_{mem} is the signature for heaps, with the usual memory access and single memory mutation functions. Σ_{set} is the theory of finite sets of addresses. The signature Σ_{reach} contains predicates to check reachability of addresses using paths at different levels. Finally, Σ_{bridge} contains auxiliary functions and predicates to manipulate and inspect paths as well as a native predicate for the skiplist memory shape.

4 Decidability of TSL

Fig. 5 shows a decision procedure for the satisfiability problem of TSL formulas, by a reduction to satisfiability of quantifier-free $\mathsf{TSL_K}$ formulas and quantifier-free Presburger arithmetic formulas. We start from a TSL formula φ in disjunctive normal form: $\varphi_1 \vee \cdots \vee \varphi_n$ so the procedure only needs to check the satisfiability of a conjunction of TSL literals φ_i. The rest of this section describes the decision procedure and proves its correctness.

A flat literal is of the form $x = y$, $x \neq y$, $x = f(y_1, \ldots, y_n)$, $p(y_1, \ldots, y_n)$ or $\neg p(y_1, \ldots, y_n)$, where $x, y, y_1, \ldots, y_n$ are variables, f is a function symbol and p is a predicate symbol defined in the signature of TSL. We first identify a set of normalized literals. All other literals can be converted into normalized literals.

Definition 1. *A normalized TSL-literal is a flat literal of the form:*

$e_1 \neq e_2$	$a_1 \neq a_2$	$l_1 \neq l_2$
$a = null$	$c = error$	$c = rd(m, a)$
$k_1 \neq k_2$	$k_1 \preceq k_2$	$m_2 = upd(m_1, a, c)$
$c = mkcell(e, k, A, l)$	$l_1 < l_2$	$l = q$
$s = \{a\}$	$s_1 = s_2 \cup s_3$	$s_1 = s_2 \setminus s_3$
$a = A[l]$	$B = A\{l \leftarrow a\}$	
$p_1 \neq p_2$	$p = [a]$	$p_1 = rev(p_2)$
$s = path2set(p)$	$append(p_1, p_2, p_3)$	$\neg append(p_1, p_2, p_3)$
$s = addr2set(m, a, l)$	$p = getp(m, a_1, a_2, l)$	
$ordList(m, p)$		$skiplist(m, s, a_1, a_2)$

where e, e_1 and e_2 are elem*-variables; a, a_1 and a_2 are* addr*-variables; c is a* cell*-variable; m, m_1 and m_2 are* mem*-variables; p, p_1, p_2 and p_3 are* path*-variables; s, s_1, s_2 and s_3 are* set*-variables; A and B* array*-variables; k, k_1 and k_2 are* ord*-variables and l, l_1 and l_2 are* level*-variables, and q is an* level *constant.*

The set of non-normalized literals consists on all flat literals not given in Definition 1. For instance, $e = c.data$ can be rewritten as $\exists_{\mathsf{ord}} k\ \exists_{\mathsf{array}} A\ \exists_{\mathsf{level}} l \mid c = mkcell(e, k, A, l)$ and $reach(m, a_1, a_2, l, p)$ can be translated into the equivalent formula $a_2 \in addr2set(m, a_1, l) \wedge p = getp(m, a_1, a_2, l)$.

Lemma 1. *Every TSL-formula is equivalent to a collection of conjunctions of normalized TSL-literals.*

For example, consider the skiplist presented in Fig. 1 and the following formula ψ that we will use as a running example:

$$\psi \quad : \quad i = 0 \wedge A = rd(heap, head).arr \wedge B = A\{i \leftarrow tail\}.$$

This formula establishes that B is an array that is equal to the next pointers of node *head*, except for the lower level that now contains the address of *tail*. To check the satisfiability of this formula we first normalize it, obtaining ψ_{norm}:

$$\psi_{\text{norm}} \quad : \quad i = 0 \ \wedge \ \begin{pmatrix} c = rd(heap, head) \wedge \\ c = mkcell(e, k, A, l) \wedge \\ l = 3 \end{pmatrix} \ \wedge \ B = A\{i \leftarrow tail\}.$$

4.1 STEP 1: Sanitation The decision procedure begins with STEP 1 by sanitizing the normalized collection of literals received as input.

Definition 2 (Sanitized). *A conjunction of normalized literals is sanitized if for every literal $B = A\{l \leftarrow a\}$ there is a literal of the form $l_{new} = l + 1$, where l_{new} is a newly introduced variable if necessary.*

The fresh level variables in sanitized formulas will be later used in the proof of Theorem 1 below to construct a proper model by replicating level l_{new} instead of level l. In turn, sanitation allows to show the existence of models with constants from models of sub-formulas without constants. Sanitizing a formula does not affect its satisfiability because it only adds an arithmetic constraint ($l_{new} = l+1$) for a fresh new variable l_{new}. Hence, a model of φ (the sanitized formula) is a

model for φ_{in} (the input formula), and from a model of φ_{in} one can immediately build a model of φ by computing the values of the variables l_{new}. Considering again our example, after sanitizing ψ_{norm} we obtain ψ_{sanit}:

$$\psi_{\text{sanit}} : \psi_{\text{norm}} \ \wedge \ l_{new} = i + 1.$$

4.2 STEP 2: Order arrangements, and STEP 3: Split In a given model of a formula, every level variable is assigned a natural number. Hence, every two variables are either assigned the same value or their values are ordered. We call these order predicates an *order arrangement*. Since there is a finite number of level variables, there is a finite number of possible order arrangements. STEP 2 consists of guessing one order arrangement.

STEP 3 uses the order arrangement to reduce the satisfiability of a sanitized formula that follows an order arrangement into the satisfiability of a Presburger Arithmetic formula (checked in STEP 4), and the satisfiability of a sanitized formula *without constants* (checked in STEP 5). An essential element in the construction is the notion of gaps. The ability to introduce gaps in models allows to show that if a model for the formula without constants exists, then a model for the formula with constants also exists (provided the Presburger constraints are also met).

Definition 3 (Gap). *Let $\mathcal{A}$ be a model of φ. We say that $n \in \mathbb{N}$ is a* gap *in $\mathcal{A}$ if there are variables l_1, l_2 in $V_{\mathsf{level}}(\varphi)$ such that $l_1^{\mathcal{A}} < n < l_2^{\mathcal{A}}$, but there is no l in $V_{\mathsf{level}}(\varphi)$ with $l^{\mathcal{A}} = n$.*

Consider ψ_{sanit} for which $V_{\mathsf{level}}(\psi_{\text{sanit}}) = \{i, l_{new}, l\}$. A model $\mathcal{A}_\psi$ that interprets variables i, l_{new} and l as 0, 1 and 3 respectively has a gap at 2. A gap-less model is a model without gaps, either between two level variables or above any level variable.

Definition 4 (Gap-less model). *A model $\mathcal{A}$ of φ is a gap-less model whenever it has no gaps, and for every array C in $\mathsf{array}^{\mathcal{A}}$ and level $n > l^{\mathcal{A}}$ for all $l \in V_{\mathsf{level}}(\varphi)$, $C(n) = null$.*

The following intermediate definition and lemma greatly simplify subsequent constructions by relating the satisfaction of literals between two models that agree on most sorts and the connectivity of relevant levels.

Definition 5. *Two interpretations $\mathcal{A}$ and $\mathcal{B}$ of a formula φ* agree *on sorts σ whenever $\mathcal{A}_\sigma = \mathcal{B}_\sigma$ and*
(i) for every $v \in V_\sigma(\varphi)$, $v^{\mathcal{A}} = v^{\mathcal{B}}$,
(ii) for every function symbol f with domain and codomain from sorts in σ, $f^{\mathcal{A}} = f^{\mathcal{B}}$ and for every predicate symbol with domain in σ, $P^{\mathcal{A}}$ iff $P^{\mathcal{B}}$.

Lemma 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two interpretations of a sanitized formula φ that agree on $\sigma : \{\mathsf{addr}, \mathsf{elem}, \mathsf{ord}, \mathsf{path}, \mathsf{set}\}$, and such that for every $l \in V_{\mathsf{level}}(\varphi)$, $m \in V_{\mathsf{mem}}(\varphi)$, and $a \in \mathsf{addr}^{\mathcal{A}}$: $m^{\mathcal{A}}(a).arr^{\mathcal{A}}(l^{\mathcal{A}}) = m^{\mathcal{B}}(a).arr^{\mathcal{B}}(l^{\mathcal{B}})$. It follows that $reach^{\mathcal{A}}(m^{\mathcal{A}}, a^{\mathcal{A}}_{init}, a^{\mathcal{A}}_{end}, l^{\mathcal{A}}, p^{\mathcal{A}})$ if and only if $reach^{\mathcal{B}}(m^{\mathcal{B}}, a^{\mathcal{B}}_{init}, a^{\mathcal{B}}_{end}, l^{\mathcal{B}}, p^{\mathcal{B}})$.*

We show now that if a sanitized formula without constants, as the one obtained after the split in STEP 3, has a model then it has a model without gaps.

Lemma 3 (Gap-reduction). *Let $\mathcal{A}$ be a model of a sanitized formula φ without constants, and let $\mathcal{A}$ have a gap at n. Then, there is a model $\mathcal{B}$ of φ such that, for every $l \in V_{\mathsf{level}}(\varphi)$: $l^{\mathcal{B}} = l^{\mathcal{A}} - 1$ if $l^{\mathcal{A}} > n$, and $l^{\mathcal{B}} = l^{\mathcal{A}}$ if $l^{\mathcal{A}} < n$. The number of gaps in $\mathcal{B}$ is one less than in $\mathcal{A}$.*

Proof. (Sketch) We show here the construction of the model and leave the exhaustive case analysis of each literal for the appendix. Let $\mathcal{A}$ be a model of φ with a gap at n. We build a model $\mathcal{B}$ with the condition in the lemma as follows. $\mathcal{B}$ agrees with $\mathcal{A}$ on $\mathsf{addr}, \mathsf{elem}, \mathsf{ord}, \mathsf{path}, \mathsf{set}$. In particular, $v^{\mathcal{B}} = v^{\mathcal{A}}$ for variables of these sorts. For the other sorts we let $\mathcal{B}_\sigma = \mathcal{A}_\sigma$ for $\sigma = \mathsf{level}, \mathsf{array}, \mathsf{cell}, \mathsf{mem}$. We define the following transformation maps:

$$\beta_{\mathsf{level}}(j) = \begin{cases} j & \text{if } j < n \\ j-1 & \text{otherwise} \end{cases} \qquad \beta_{\mathsf{array}}(A)(i) = \begin{cases} A(i) & \text{if } i < n \\ A(i+1) & \text{if } i \geq n \end{cases}$$

$$\beta_{\mathsf{cell}}((e,k,A,l)) = (e,k,\beta_{\mathsf{array}}(A),\beta_{\mathsf{level}}(l)) \qquad \beta_{\mathsf{mem}}(m)(a) = \beta_{\mathsf{cell}}(m(a))$$

Now we are ready to define the valuations of variables $l : \mathsf{level}$, $A : \mathsf{array}$, $c : \mathsf{cell}$ and $m : \mathsf{mem}$:

$$l^{\mathcal{B}} = \beta_{\mathsf{level}}(l^{\mathcal{A}}) \qquad A^{\mathcal{B}} = \beta_{\mathsf{array}}(A^{\mathcal{A}}) \qquad c^{\mathcal{B}} = \beta_{\mathsf{cell}}(c^{\mathcal{A}}) \qquad m^{\mathcal{B}} = \beta_{\mathsf{mem}}(m^{\mathcal{A}})$$

The interpretation of all functions and predicates is preserved from $\mathcal{A}$. An exhaustive case analysis on the normalized literals allows to show that $\mathcal{B}$ is indeed a model of φ. □

For instance, considering formula ψ_{sanit} and model $\mathcal{A}_\psi$, we can construct model $\mathcal{B}_\psi$ reducing one gap from $\mathcal{A}_\psi$ by stating that $i^{\mathcal{B}_\psi} = i^{\mathcal{A}_\psi}$, ${l_{new}}^{\mathcal{B}_\psi} = {l_{new}}^{\mathcal{A}_\psi}$ and $l^{\mathcal{B}_\psi} = 2$, and completely ignoring arrays in model $\mathcal{A}_\psi$ at level 2.

Lemma 4 (Top-reduction). *Let $\mathcal{A}$ be a model of φ, and n a level such that $n > l^{\mathcal{A}}$ for all $l \in V_{\mathsf{level}}(\varphi)$ and $A \in \mathsf{array}^{\mathcal{A}}$ be such that $A(n) \neq null$. Then the interpretation $\mathcal{B}$ obtained by replacing $A(n) = null$ is also a model of φ.*

Proof. By a simple case analysis on the literals of φ, using Lemma 2. □

Corollary 1. *Let φ be a sanitized formula without constants. Then, φ has a model if and only if φ has a gapless model.*

STEP 2 in the decision procedure guesses an order arrangement of level variables from the sanitized formula φ. Informally, an order arrangement is a total order between the equivalence classes of level variables.

Definition 6 (Order Arrangement). *Given a sanitized formula φ, an order arrangement is a collection of literals containing, for every pair of level variables $l_1, l_2 \in V_{\mathsf{level}}(\varphi)$, exactly one of: $(l_1 = l_2)$, $(l_1 < l_2)$, or $(l_2 < l_1)$.*

For instance, an order arrangement of ψ_{sanit} is $\{i < l_{new}, i < l, l_{new} < l\}$. As depicted in Fig. 5 (right), STEP 3 of the decision procedure splits the sanitized formula φ into φ^{PA}, which contains precisely all those literals in the theory of arithmetic Σ_{level}, and φ^{NC} containing all literals from φ except those involving constants ($l = q$). Clearly, φ is equivalent to $\varphi^{\text{NC}} \wedge \varphi^{\text{PA}}$. In our case, ψ_{sanit} is split into ψ^{PA} and ψ^{NC}:

$$\begin{array}{lcl} \psi^{\text{PA}} & : & i = 0 \wedge l = 3 \wedge l_{new} = i + 1 \\ \psi^{\text{NC}} & : & \begin{pmatrix} c = rd(heap, head) \wedge \\ c = mkcell(e, k, A, l) \end{pmatrix} \wedge B = A\{i \leftarrow tail\} \wedge l_{new} = i + 1. \end{array}$$

For a given formula there is only a finite collection of order arrangements satisfying φ^{PA}. We use $arr(\varphi^{\text{PA}})$ for the set of order arrangements of variables satisfying φ^{PA}. A model of φ^{PA} is characterized by a map $f : V_{\mathsf{level}}(\varphi) \to \mathbb{N}$ assigning a natural number to each level variable. In the case of ψ^{PA}, f maps i, l_{new} and l to 0, 1 and 3 respectively. Also, for every model f of φ^{PA} there is a unique order arrangement $\alpha \in arr(\varphi^{\text{PA}})$ for which $f \vDash \alpha$. STEP 4 consists of checking whether there is a model of φ^{PA} that corresponds to a given order arrangement α by simply checking the satisfiability of the Presburger arithmetic formula $(\varphi^{\text{PA}} \wedge \alpha)$.

We are now ready to show that the guess in STEP 2 and the split in STEP 3 preserve satisfiability. Theorem 1 below allows to reduce the satisfiability of φ to the satisfiability of a Presburger Arithmetic formula and the satisfiability of a TSL formula without constants. We show in the next section how to decide this fragment of TSL.

Theorem 1. *A sanitized TSL formula φ is satisfiable if and only if for some order arrangement α, both $(\varphi^{PA} \wedge \alpha)$ and $(\varphi^{NC} \wedge \alpha)$ are satisfiable.*

4.3 STEP 4: Presburger Constraints The formula φ^{PA} contains only literals of the form $l_1 = q$, $l_1 \neq l_2$, $l_1 = l_2 + 1$, and $l_1 < l_2$ for integer variables l_1 and l_2 and integer constant q. The satisfiability of this kind of formulas can be easily decided with off-the-shelf SMT solvers. If φ^{PA} is unsatisfiable then the original formula (for the guessed order arrangement) is also unsatisfiable.

4.1 STEP 5: Deciding Satisfiability of Formulas Without Constants

We show here the correctness of the reduction of the satisfiability of a sanitized formula without constants to the satisfiability of a formula in the decidable theory $\mathsf{TSL_K}$ (STEP 5). That is, we detail how to generate from a sanitized formula without constants ψ (formula $(\varphi \wedge \alpha)$ in Fig. 5) an equisatisfiable $\mathsf{TSL_K}$ formula $\ulcorner\psi\urcorner$ for a finite value K computed from the formula. The bound is $\mathsf{K} = |V_{\mathsf{level}}(\psi)|$. This bound limits the number of levels required in the reasoning. We use $[\mathsf{K}]$ as a short for the set $0 \ldots \mathsf{K} - 1$. For ψ_{sanit}, we have $\mathsf{K} = 3$ and thus we construct a formula in $\mathsf{TSL_3}$.

The translation from ψ into $\ulcorner\psi\urcorner$ works as follows. For every variable A of sort array appearing in some literal in ψ we introduce K fresh new variables $v_{A[0]}, \ldots, v_{A[\mathsf{K}-1]}$ of sort addr. These variables correspond to the addresses from A that the decision procedure for $\mathsf{TSL_K}$ needs to reason about. All literals from ψ are left unchanged in $\ulcorner\psi\urcorner$ except $(c = \mathit{mkcell}(e, k, A, l))$, $(a = A[l])$, $(B = A\{l \leftarrow a\})$, $B = A$ and $\mathit{skiplist}(m, s, a_1, a_2)$ that are changed as follows:

- $c = \mathit{mkcell}(e, k, A, l)$ is transformed into $c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]})$.
- $a = A[l]$ gets translated into: $\bigwedge_{i=0\ldots\mathsf{K}-1} l = i \to a = v_{A[i]}$.
- $B = A\{l \leftarrow a\}$ is translated into:

$$\Big(\bigwedge_{i=0\ldots\mathsf{K}-1} l = i \to a = v_{B[i]}\Big) \wedge \Big(\bigwedge_{j=0\ldots\mathsf{K}-1} l \neq j \to v_{B[j]} = v_{A[j]}\Big) \qquad (1)$$

- $\mathit{skiplist}(m, r, a_1, a_2)$ gets translated into:

$$\begin{array}{ll} & \mathit{ordList}(m, \mathit{getp}(m, a_1, a_2, 0)) \quad \wedge \quad r = \mathit{path2set}(\mathit{getp}(m, a_1, a_2, 0)) \wedge \\ \bigwedge_{i\in 0\ldots\mathsf{K}-1} & \mathit{rd}(m, a_2).\mathit{arr}[i] = \mathit{null} \qquad \wedge \\ \bigwedge_{i\in 0\ldots\mathsf{K}-2} & \mathit{path2set}(\mathit{getp}(m, a_1, a_2, i+1)) \subseteq \mathit{path2set}(\mathit{getp}(m, a_1, a_2, i)) \end{array} \qquad (2)$$

Note that the formula $\ulcorner\varphi\urcorner$ obtained using this translation belongs to the theory $\mathsf{TSL_K}$. For instance,

$$\ulcorner\psi^{\mathrm{NC}}\urcorner : \begin{bmatrix} i = 0 \to \mathit{tail} = v_{B[0]} & \wedge\, i = 1 \to \mathit{tail} = v_{B[1]} & \wedge\, i = 2 \to \mathit{tail} = v_{B[2]} & \wedge \\ i \neq 0 \to v_{B[0]} = v_{A[0]} & \wedge\, i \neq 1 \to v_{B[1]} = v_{A[1]} & \wedge\, i \neq 2 \to v_{B[2]} = v_{A[2]} & \wedge \\ c = \mathit{rd}(\mathit{heap}, \mathit{head}) & \wedge\, c = \mathit{mkcell}(e, k, v_{A[0]}, v_{A[1]}, v_{A[2]}) & \wedge\, l_{new} = i + 1 & \end{bmatrix}$$

The following lemma establishes the correctness of the translation.

Lemma 5. *Let ψ be a sanitized TSL formula with no constants. Then, ψ is satisfiable if and only if $\ulcorner\psi\urcorner$ is also satisfiable.*

The main result of this paper is the following decidability theorem, which follows immediately from Lemma 5, Theorem 1 and the fact that every formula can be normalized and sanitized.

Theorem 2. *The satisfiability problem of* (QF) *TSL-formulas is decidable.*

5 Example: Skiplist Preservation

We sketch the proof that the implementation given in Fig. 2 preserves the skiplist shape property. This is a safety property, and can be proved using invariance: the data structure initially has a skiplist shape and all transitions preserve this shape.

This invariance proof is automatically decomposed in the following verification conditions:

$$(\text{INI}) : \Theta \to \mathit{skiplist} \qquad\qquad (\text{CON}) : \bigwedge_{i\in 1\ldots 79} \mathit{skiplist} \wedge \tau_i \to \mathit{skiplist}'$$

where Θ denotes the initial condition and τ_i is the transition relation $\tau_i(V, V')$ corresponding to program line i, relating variables in the pre-state (V) with variables in the post-state (V'). Finally, *skiplist* and *skiplist'* are short notation for $\mathit{skiplist}(\mathit{heap}, r, \mathit{maxLevel}, \mathit{head}, \mathit{tail})$ and $\mathit{skiplist}(\mathit{heap}', r', \mathit{maxLevel}', \mathit{head}', \mathit{tail}')$ respectively. All VCs discharged are quantifier-free TSL formulas and thus are verifiable using our decision procedure. We use a single value to denote the key and value of a cell, hence a cell (v, A, l) represents (v, v, A, l) and $rd(c)$ as a short for $rd(\mathit{heap}, c)$. Condition (INI) is easy to verify, from initial condition Θ:

$$\Theta \mathrel{\hat{=}} \begin{bmatrix} rd(\mathit{head}) = c_h \wedge c_h = (-\infty, A_h, 0) \wedge A_h[0] = \mathit{tail} \wedge \ \mathit{maxLevel} = 0 \ \wedge \\ rd(\mathit{tail}) = c_t \ \wedge \ c_t = (+\infty, A_t, 0) \ \wedge A_t[0] = \mathit{null} \wedge r = \{\mathit{head}, \mathit{tail}\} \end{bmatrix}$$

To prove the validity of (CON), we negate it and show that $\mathit{skiplist} \wedge \tau_i \wedge \neg\mathit{skiplist}'$ is unsatisfiable. As shown above, $\neg\mathit{skiplist}'$ is normalized into five disjuncts. Two of them are: (NSL1) $\big(\neg\mathit{ordList}(m, \mathit{getp}(\mathit{heap}, \mathit{head}, \mathit{tail}, 0))\big)$; and (NSL4) $\big(a \in \mathit{reg} \wedge rd(\mathit{heap}, a).\mathit{level} > \mathit{maxLevel}\big)$.

Consider (NSL1). The only offending transition that could satisfy the negation of the VC is τ_{36}, which connects a new cell to the skiplist. We can automatically prove that this transition preserves the skiplist order using the following supporting invariants:

$$\varphi_{\text{next}} \mathrel{\hat{=}} \begin{pmatrix} pc = 21 \to \mathit{curr} = rd(\mathit{pred}).\mathit{arr}[\mathit{maxLevel}] \wedge \\ pc = 22..25, 27..29, 60..63, 65..70 \to \mathit{curr} = rd(\mathit{pred}).\mathit{arr}[i] \end{pmatrix}$$

$$\varphi_{\text{predLess}} \mathrel{\hat{=}} pc = 20..40, 59..79 \to \begin{pmatrix} rd(\mathit{pred}).\mathit{val} < v \wedge \\ rd(\mathit{pred}).\mathit{val} < rd(\mathit{tail}).\mathit{val} \end{pmatrix}$$

$$\varphi_{\text{ord}(j)} \mathrel{\hat{=}} \begin{pmatrix} (pc = 22..38, 60..70, 73..75 \wedge i < j \leq \mathit{maxLevel}) \vee \\ (pc = 71..72 \qquad\qquad\qquad \wedge\, 0 \leq j \leq \mathit{maxLevel}) \end{pmatrix} \to \begin{pmatrix} rd(\mathit{upd}[j]).\mathit{val} < v \wedge \\ rd(rd(\mathit{upd}[j]).\mathit{arr}[j]).\mathit{val} \geq v \end{pmatrix}$$

where pc denotes the program counter. We use $(pc = a..b)$ to denote $(pc = a \vee \cdots \vee pc = b)$. Invariant φ_{next} establishes that *curr* points to the next cell pointed by *pred* at level i. Invariant $\varphi_{\text{predLess}}$ says that the value pointed by *pred* is always strictly lower than the value we are inserting or removing, and the value pointed by *tail*. Finally, $\varphi_{\text{ord}(j)}$ establishes that when inside the loops, array *upd* at level j points to the last cell whose value is strictly lower than the value to be inserted or removed. This way, when taking τ_{36}, the decision procedure can show that the order of elements in the list is preserved.

Checking (NSL4) is even simpler, requiring only the following invariant:

$$\varphi_{\text{bound}} \mathrel{\hat{=}} (pc = 19..40 \to \mathit{lvl} \leq \mathit{maxLevel}) \wedge (pc = 34..40 \to rd(x).\mathit{level} = \mathit{lvl})$$

A similar approach is followed for all other cases of $\neg\mathit{skiplist}'$.

6 Conclusion and Future Work

In this paper we have presented TSL, a theory of skiplists of arbitrary many levels, useful for automatically prove the VCs generated during the verification of skiplist implementations. TSL is capable of reasoning about memory, cells, pointers, regions and reachability, ordered lists and sublists, allowing the description of the skiplist property, and the representation of memory modifications introduced by the execution of program statements. The main novelty of TSL is that it is not limited to skiplists of a limited height.

We showed that TSL is decidable by reducing its satisfiability problem to TSL_K [12] (a decidable theory capable of reasoning about skiplists of bounded levels) and we illustrated such reduction by some examples. Our reduction allows to restrict the reasoning to only the levels being explicitly accessed in the (sanitized) formula.

Future work also includes the temporal verification of sequential and concurrent skiplists implementations, including industrial implementations like in the `java.concurrent` standard library. We are currently implementing our decision procedure on top of off-the-shelf SMT solvers such as Yices and Z3. This implementation so far provides a very promising performance for the automation of skiplist proofs. However, reports on this empirical evaluation is future work.

References

1. A. Bouajjani, C. Dragoi, C. Enea, and M. Sighireanu. A logic-based framework for reasoning about composite data structures. In *CONCUR'09*, pages 178–195, 2009.
2. A. Browne, Z. Manna, and H. B. Sipma. Generalized verification diagrams. In *Proc. of FSTTCS'95*, volume 1206 of *LNCS*, pages 484–498. Springer, 1995.
3. V. Kuncak, H. H. Nguyen, and M. C. Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *CADE'05*, pages 260–277, 2005.
4. S. K. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *Proc. of POPL'08*, pages 171–182. ACM, 2008.
5. P. Madhusudan, G. Parlato, and X. Qiu. Decidable logics combining heap structures and data. In *Proc. of POPL'11*, pages 611–622. ACM, 2011.
6. Z. Manna and A. Pnueli. *Temporal Verif. of Reactive Sys.* Springer, 1995.
7. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
8. W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.
9. S. Ranise and C. G. Zarba. A theory of singly-linked lists and its extensible decision procedure. In *Proc. of SEFM 2006*. IEEE CS Press, 2006.
10. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74. IEEE CS Press, 2002.
11. A. Sánchez and C. Sánchez. Decision procedures for the temporal verification of concurrent lists. In *Proc. of ICFEM'10*, volume 6447 of *LNCS*, pages 74–89, 2010.
12. A. Sánchez and C. Sánchez. A theory of skiplists with applications to the verif. of concurrent datatypes. In *Proc. of NFM 2011*, volume 6617 of *LNCS*, 2011.
13. G. Yorsh, A. M. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. In *FOSSACS'06*, pages 94–110, 2006.

A Missing Proofs

Lemma 1. *Every TSL-formula is equivalent to a collection of conjunctions of normalized TSL-literals.*

Proof. By case analysis on non-normalized literals. For illustration purpose we show some interesting cases only. For instance, $\neg ordList(m, p)$ is equivalent to:

$$
\begin{array}{ll}
(\exists l_1, l_2, zero : \mathsf{level})\ (\exists a_1, a_2 : \mathsf{addr})\ (\exists c_1, c_2 : \mathsf{cell}) & \\
(\exists e_1, e_2 : \mathsf{elem})\ (\exists k_1, k_2 : \mathsf{ord})\ (\exists A_1, A_2 : \mathsf{array}) & \\
\quad a_1 \in path2set(p) \wedge a_2 \in path2set(p) \wedge zero = 0 \quad \wedge & (3) \\
\quad c_1 = rd(m, a_1) \wedge c_1 = mkcell(e_1, k_1, A_1, l_1) \quad \wedge & (4) \\
\quad a_2 = A_1[zero] \wedge c_2 = rd(m, a_2) \wedge c_2 = mkcell(e_2, k_2, A_2, l_2) \quad \wedge & (5) \\
\quad k_2 \preceq k_1 \wedge k_2 \neq k_1 & (6)
\end{array}
$$

Conjunct (3) establishes that there are two witness addresses a_1 and a_2 in path p. Literal (4) captures that c_1 is the cell at which a_1 is mapped in memory m. Conjunct (5) captures that c_2 is the cell next to c_1 on memory m, following pointers at level 0. That is, c_2 immediately follows c_1 in heap m. Finally, (6) establishes that the key of c_1 is strictly greater that the key of c_2, violating the order of the list.

As another example, consider literal $\neg skiplist(m, r, l, a_i, a_e)$. Based on the interpretation given in Fig. 4, this literal is equivalent to the following:

$$
\begin{array}{ll}
\left[(\exists p : \mathsf{path})\ p = getp(m, a_i, a_e, 0) \wedge \neg ordList(m, p) \right] \vee & \text{(NSL1)} \\
\left[(\exists p : \mathsf{path})(\exists s : \mathsf{set})\ p = getp(m, a_i, a_e, 0) \wedge s = path2set(p) \wedge r \neq s \right] \vee & \text{(NSL2)} \\
\left[l < 0 \right] \vee & \text{(NSL3)} \\
\left[\begin{array}{l} (\exists a : \mathsf{addr})(\exists e : \mathsf{elem})(\exists k : \mathsf{ord})(\exists A : \mathsf{array})(\exists \tilde{l} : \mathsf{level})(\exists c : \mathsf{cell}) \\ \quad a \in r \wedge c = rd(m, a) \wedge c = mkcell(e, k, A, \tilde{l}) \wedge l < \tilde{l} \end{array} \right] \vee & \text{(NSL4)}
\end{array}
$$

$$
\begin{array}{ll}
\left[\begin{array}{l} (\exists a : \mathsf{addr})(\exists e : \mathsf{elem})(\exists k : \mathsf{ord})(\exists A : \mathsf{array})(\exists l_1, l_2 : \mathsf{level}) \\ (\exists c : \mathsf{cell}) \\ \quad l \neq 0 \wedge 0 \leq l_2 \wedge l_2 \leq l_1 \wedge \\ \quad c = rd(m, a_e) \wedge c = mkcell(e, k, A, l_1) \wedge a = A[l_2] \wedge a \neq null \end{array} \right] \vee & \text{(NSL5)} \\
\left[\begin{array}{l} (\exists l_1, l_2 : \mathsf{level})(\exists p_1, p_2 : \mathsf{path})(\exists s_1, s_2 : \mathsf{set}) \\ \quad l \neq 0 \wedge 0 \leq l_1 \wedge l_1 < l \wedge l_2 = s(l_1) \wedge \\ \quad p_1 = getp(m, a_i, a_e, l_1) \wedge p_2 = getp(m, a_i, a_e, l_2) \wedge \\ \quad s_1 = path2set(p_1) \wedge s_2 = path2set(p_2) \wedge s_1 \not\subseteq s_2 \end{array} \right] & \text{(NSL6)}
\end{array}
$$

Literals such as $a \in r$, $\neg ordList(m, p)$ and $l < 0$ are not normalized, but we leave them in the previous formulas for simplicity. □

Lemma 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two interpretations of a sanitized formula φ that agree on $\sigma : \{\mathsf{addr}, \mathsf{elem}, \mathsf{ord}, \mathsf{path}, \mathsf{set}\}$, and such that for every $l \in V_{\mathsf{level}}(\varphi)$, $m \in V_{\mathsf{mem}}(\varphi)$, and $a \in \mathsf{addr}^{\mathcal{A}}$: $m^{\mathcal{A}}(a).arr^{\mathcal{A}}(l^{\mathcal{A}}) = m^{\mathcal{B}}(a).arr^{\mathcal{B}}(l^{\mathcal{B}})$. It follows that $reach^{\mathcal{A}}(m^{\mathcal{A}}, a^{\mathcal{A}}_{init}, a^{\mathcal{A}}_{end}, l^{\mathcal{A}}, p^{\mathcal{A}})$ if and only if $reach^{\mathcal{B}}(m^{\mathcal{B}}, a^{\mathcal{B}}_{init}, a^{\mathcal{B}}_{end}, l^{\mathcal{B}}, p^{\mathcal{B}})$.*

Proof. Let $\mathcal{A}$ and $\mathcal{B}$ be two interpretations of φ satisfying the conditions in the statement of Lemma 2, and assume $reach^{\mathcal{A}}(m^{\mathcal{A}}, a^{\mathcal{A}}_{init}, a^{\mathcal{A}}_{end}, l^{\mathcal{A}}, p^{\mathcal{A}})$ holds for some $a_{init}, a_{end} \in V_{\mathsf{addr}}(\varphi)$, $m \in V_{\mathsf{mem}}(\varphi)$, $p \in V_{\mathsf{path}}(\varphi)$. Note that, by assumption $a^{\mathcal{A}}_{init} = a^{\mathcal{B}}_{init}$, $a^{\mathcal{A}}_{end} = a^{\mathcal{B}}_{end}$ and $p^{\mathcal{A}} = p^{\mathcal{B}}$. We consider the cases for $p^{\mathcal{A}}$:

- If $p^{\mathcal{A}} = \epsilon$ then $a^{\mathcal{A}}_{init} = a^{\mathcal{A}}_{end}$. Consequently, $p^{\mathcal{B}} = \epsilon$ and $a^{\mathcal{B}}_{init} = a^{\mathcal{B}}_{end}$, so for interpretation $\mathcal{B}$, the predicate $reach^{\mathcal{A}}(m^{\mathcal{B}}, a^{\mathcal{B}}_{init}, a^{\mathcal{B}}_{end}, l^{\mathcal{B}}, p^{\mathcal{B}})$ also holds.
- The other case is: $p = [a_1 \dots a_n]$ with $a_1 = a_{init}$ and $m^{\mathcal{A}}(a_n).arr^{\mathcal{A}}(l^{\mathcal{A}}) = a_{end}$, and for every $r < n$, $m^{\mathcal{A}}(a_r).arr^{\mathcal{A}}(l^{\mathcal{A}}) = a_{r+1}$. It follows, by (**??**) that $m^{\mathcal{B}}(a_n).arr^{\mathcal{B}}(l^{\mathcal{B}}) = a_{end}$, and for every $r < n$, $m^{\mathcal{B}}(a_r).arr^{\mathcal{B}}(l^{\mathcal{B}}) = a_{r+1}$. Hence, $reach^{\mathcal{A}}(m^{\mathcal{B}}, a^{\mathcal{B}}_{init}, a^{\mathcal{B}}_{end}, l^{\mathcal{B}}, p^{\mathcal{B}})$.

The other direction follows similarly. □

Lemma 3 (Gap-reduction). *If there is a model $\mathcal{A}$ of φ with a gap at n, then there is a model $\mathcal{B}$ of φ such that, for every $l \in V_{\mathsf{level}}(\varphi)$, we let*

$$l^{\mathcal{B}} = \begin{cases} l^{\mathcal{A}} & \textit{if } l^{\mathcal{A}} < n \\ l^{\mathcal{A}} - 1 & \textit{if } l^{\mathcal{A}} > n \end{cases}$$

The number of gaps in $\mathcal{B}$ is one less than in $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be a model of φ with a gap at n. We build a model $\mathcal{B}$ with the condition in the lemma as follows. $\mathcal{B}$ agrees with $\mathcal{A}$ on $\mathsf{addr}, \mathsf{elem}, \mathsf{ord}, \mathsf{path}, \mathsf{set}$. In particular, $v^{\mathcal{B}} = v^{\mathcal{A}}$ for variales of these sorts. For the other sorts we let $\mathcal{B}_\sigma = \mathcal{A}_\sigma$ for $\sigma = \mathsf{level}, \mathsf{array}, \mathsf{cell}, \mathsf{mem}$. We define transformation maps for elements of the corresponding domains as follows:

$$\beta_{\mathsf{level}}(j) = \begin{cases} j & \text{if } j < n \\ j - 1 & \text{otherwise} \end{cases} \qquad \beta_{\mathsf{array}}(A)(i) = \begin{cases} A(i) & \text{if } i < n \\ A(i+1) & \text{if } i \geq n \end{cases}$$

$$\beta_{\mathsf{cell}}((e, k, A, l)) = (e, k, \beta_{\mathsf{array}}(A), \beta_{\mathsf{level}}(l)) \qquad \beta_{\mathsf{mem}}(m)(a) = \beta_{\mathsf{cell}}(m(a))$$

Now we are ready to define the valuations of variables $l : \mathsf{level}$, $A : \mathsf{array}$, $c : \mathsf{cell}$ and $m : \mathsf{mem}$:

$$l^{\mathcal{B}} = \beta_{\mathsf{level}}(l^{\mathcal{A}}) \qquad A^{\mathcal{B}} = \beta_{\mathsf{array}}(A^{\mathcal{A}}) \qquad c^{\mathcal{B}} = \beta_{\mathsf{cell}}(c^{\mathcal{A}}) \qquad m^{\mathcal{B}} = \beta_{\mathsf{mem}}(m^{\mathcal{A}})$$

The interpretation of all functions and predicates is preserved from $\mathcal{A}$.

The next step is to show that $\mathcal{B}$ is indeed a model of φ. All literals of the following form hold in $\mathcal{B}$ because if they hold in $\mathcal{A}$, because the valuations and in-

terpretations of functions and predicates of the correspondig sorts are preserved:

$e_1 \neq e_2$	$a_1 \neq a_2$	$l_1 \neq l_2$
$a = null$	$c = error$	
$k_1 \neq k_2$	$k_1 \preceq k_2$	
	$l_1 < l_2$	$l = q$
$s = \{a\}$	$s_1 = s_2 \cup s_3$	$s_1 = s_2 \setminus s_3$
$p_1 \neq p_2$	$p = [a]$	$p_1 = rev(p_2)$
$s = path2set(p)$	$append(p_1, p_2, p_3)$	$\neg append(p_1, p_2, p_3)$
		$ordList(m, p)$

A simple argument shows that literals of the form $c = rd(m, a)$ and $m_2 = upd(m_1, a, c)$ hold in $\mathcal{B}$ if they do in $\mathcal{A}$, because the same transformations are performed on both sides of the equation. The remaining literals are:

- $c = mkcell(e, k, A, l)$: Assuming $c^{\mathcal{A}} = mkcell(e^{\mathcal{A}}, k^{\mathcal{A}}, A^{\mathcal{A}}, l^{\mathcal{A}})$,

  $$mkcell(e^{\mathcal{B}}, k^{\mathcal{B}}, A^{\mathcal{B}}, l^{\mathcal{B}}) = mkcell(e^{\mathcal{A}}, k^{\mathcal{A}}, \beta_{\mathsf{array}}(A^{\mathcal{A}}), \beta_{\mathsf{level}}(l^{\mathcal{A}})) = \beta_{\mathsf{cell}}(c^{\mathcal{A}}) = c^{\mathcal{B}}$$

- $a = A[l]$. Assume $a^{\mathcal{A}} = A^{\mathcal{A}}[l^{\mathcal{A}}]$. There are two cases for $l^{\mathcal{A}}$. First, $l^{\mathcal{A}} < n$. Then,

  $$A^{\mathcal{B}}[l^{\mathcal{B}}] = A^{\mathcal{A}}[l^{\mathcal{A}}] = a^{\mathcal{A}} = a^{\mathcal{B}}$$

  Second, $l^{\mathcal{A}} > n$. Then,

  $$A^{\mathcal{B}}[l^{\mathcal{B}}] = A^{\mathcal{A}}[(l^{\mathcal{A}} - 1) + 1] = A^{\mathcal{A}}[l^{\mathcal{A}}] = a^{\mathcal{A}} = a^{\mathcal{B}}$$

- $B = A\{l \leftarrow a\}$. We assume $B^{\mathcal{A}} = A^{\mathcal{A}}\{l^{\mathcal{A}} \leftarrow a^{\mathcal{A}}\}$. Consider an arbitary $m \in \mathbb{N}$. If $m = l^{\mathcal{B}}$ then

  $$(A^{\mathcal{B}}\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) = (\beta_{\mathsf{array}}(A^{\mathcal{A}})\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) = a^{\mathcal{B}}$$

  If $m \neq l^{\mathcal{B}}$ and $m < n$ then

  $$\begin{aligned}(A^{\mathcal{B}}\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) &= (\beta_{\mathsf{array}}(A^{\mathcal{A}})\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) = \\ &= (\beta_{\mathsf{array}}(A^{\mathcal{A}}))(m) \qquad = A^{\mathcal{A}}(m) = B^{\mathcal{A}}(m) = \\ &= \beta_{\mathsf{array}}(B^{\mathcal{A}}(m)) \qquad = B^{\mathcal{B}}(m)\end{aligned}$$

  Finally, the last case is $m \neq l^{\mathcal{B}}$ and $m \geq n$. In this case:

  $$\begin{aligned}(A^{\mathcal{B}}\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) &= (\beta_{\mathsf{array}}(A^{\mathcal{A}})\{l^{\mathcal{B}} \leftarrow a^{\mathcal{B}}\})(m) = \\ &= (\beta_{\mathsf{array}}(A^{\mathcal{A}}))(m) \qquad = A^{\mathcal{A}}(m+1) = B^{\mathcal{A}}(m+1) = \\ &= \beta_{\mathsf{array}}(B^{\mathcal{A}})(m) \qquad = B^{\mathcal{B}}(m)\end{aligned}$$

– $s = \mathit{addr2set}(m, a, l)$ and $p = \mathit{getp}(m, a_1, a_2, l)$. We first prove that for all variables m and l, addresses a_{init}, a_{end} and paths p, $\mathit{reach}(m^{\mathcal{A}}, a_{init}, a_{end}, l^{\mathcal{A}}, p)$ if and only if $\mathit{reach}(m^{\mathcal{B}}, a_{init}, a_{end}, l^{\mathcal{B}}, p)$. Assume $\mathit{reach}(m^{\mathcal{A}}, a_{init}, a_{end}, l^{\mathcal{A}}, p)$, then either $a_{init} = a_{end}$ and $p = \epsilon$, in which case $\mathit{reach}(m^{\mathcal{B}}, a_{init}, a_{end}, l^{\mathcal{B}}, p)$, or there is a sequence of addresses $a_1, \dots a_N$ with
  (a) $p = [a_1 \dots a_N]$
  (b) $a_1 = a_{init}$
  (c) $m^{\mathcal{A}}(a_r).\mathit{arr}^{\mathcal{A}}(l^{\mathcal{A}}) = a_{r+1}$, for $r < N$
  (d) $m^{\mathcal{A}}(a_N).\mathit{arr}^{\mathcal{A}}(l^{\mathcal{A}}) = a_{end}$
  Take an arbitrary $r < N$. Either $l^{\mathcal{A}} < n$ or $l^{\mathcal{A}} > n$ (recall that $l^{\mathcal{A}}$ is either strictly under or strictly over the gap). In either case,

$$m^{\mathcal{B}}(a_r).\mathit{arr}^{\mathcal{B}}(l^{\mathcal{B}}) = m^{\mathcal{A}}(a_r).\mathit{arr}^{\mathcal{A}}(l^{\mathcal{A}}) = a_{r+1}$$

  Also, $m^{\mathcal{B}}(a_N).\mathit{arr}^{\mathcal{B}}(l^{\mathcal{B}}) = m^{\mathcal{B}}(a_N).\mathit{arr}^{\mathcal{B}}(l^{\mathcal{B}}) = a_{end}$. Hence, conditions (a), (b), (c) and (d) hold for $\mathcal{B}$ and $\mathit{reach}(m^{\mathcal{B}}, a_{init}, a_{end}, l^{\mathcal{B}}, p)$. Informally, predicate *reach* only depends on pointers at level l which are preserved. The other direction holds similarly. From the preservation of the *reach* predicate it follows that, if $\mathit{addr2set}(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}}) = s^{\mathcal{A}}$ then

$$\begin{aligned}\mathit{addr2set}(m^{\mathcal{B}}, a^{\mathcal{B}}, l^{\mathcal{B}}) &= \{a' \mid \exists p \in \mathcal{B}_{\mathsf{path}} \,.\, (m, a, a', l, p \in \mathit{reach}^{\mathcal{B}}\} = \\ &= \{a' \mid \exists p \in \mathcal{A}_{\mathsf{path}} \,.\, (m, a, a', l, p \in \mathit{reach}^{\mathcal{A}}\} = \\ &= \mathit{addr2set}(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}}) = s^{\mathcal{A}} = s^{\mathcal{B}}\end{aligned}$$

  Finally, assume $p^{\mathcal{A}} = \mathit{getp}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, l^{\mathcal{A}})$. If $(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, l^{\mathcal{A}}, p^{\mathcal{A}}) \in \mathit{reach}^{\mathcal{A}}$ then $(m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, l^{\mathcal{B}}, p^{\mathcal{B}}) \in \mathit{reach}^{\mathcal{B}}$ and hence $p^{\mathcal{B}} = \mathit{getp}(m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, l^{\mathcal{B}})$. The other case is $\epsilon = \mathit{getp}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, l^{\mathcal{A}})$ when

$$\textit{for no path } p, \quad (m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, l^{\mathcal{A}}, p) \in \mathit{reach}^{\mathcal{A}}.$$

  but then also

$$\textit{for no path } p, \quad (m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, l^{\mathcal{B}}, p) \in \mathit{reach}^{\mathcal{B}}$$

  and then $\epsilon = \mathit{getp}(m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, l^{\mathcal{B}})$, as desired.

– $\mathit{skiplist}(m, r, l, a_1, a_2)$. We assume $\mathit{skiplist}(m^{\mathcal{A}}, r^{\mathcal{A}}, l^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}})$. This implies:
  - $\mathit{ordList}^{\mathcal{A}}(m^{\mathcal{A}}, \mathit{getp}^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. Let p be an element of $\mathcal{A}_{\mathsf{path}}$ such that $p = \mathit{getp}^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. As shown previously, $p = \mathit{getp}^{\mathcal{B}}(a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, 0))$, and then $\mathit{ordList}^{\mathcal{B}}(m^{\mathcal{B}}, \mathit{getp}^{\mathcal{B}}(a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, 0))$ holds because $\mathit{ordList}^{\mathcal{A}}(m^{\mathcal{A}}, \mathit{getp}^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$ does.
  - $r^{\mathcal{A}} = \mathit{path2set}^{\mathcal{A}}(\mathit{getp}^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. Again $r^{\mathcal{B}} = \mathit{path2set}^{\mathcal{B}}(\mathit{getp}^{\mathcal{B}}(m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, 0))$ because $\mathit{getp}^{\mathcal{B}}(m^{\mathcal{B}}, a_1^{\mathcal{B}}, a_2^{\mathcal{B}}, 0) = \mathit{getp}^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0)$.
  - $0 \leq l^{\mathcal{A}}$, which implies $0 \leq l^{\mathcal{B}}$
  - $\forall a \in r^{\mathcal{A}} \,.\, m^{\mathcal{A}}(a^{\mathcal{A}}).\mathit{max}^{\mathcal{A}} \leq l^{\mathcal{A}}$. Since $r^{\mathcal{B}} = r^{\mathcal{A}}$ and $m^{\mathcal{B}}(a) = \beta_{\mathsf{cell}}(m^{\mathcal{A}}(a))$ it is enough to consider two cases. First, $m^{\mathcal{A}}(a).\mathit{max}^{\mathcal{A}} = l^{\mathcal{A}}$, in which case $m^{\mathcal{B}}(a).\mathit{max}^{\mathcal{B}} = l^{\mathcal{B}}$. Second $m^{\mathcal{A}}(a).\mathit{max}^{\mathcal{A}} < l^{\mathcal{A}}$, in which case $m^{\mathcal{A}}(a).\mathit{max}^{\mathcal{A}} \leq l^{\mathcal{A}}$.

- If $(0 = l^{\mathcal{A}})$ then $(0 = l^{\mathcal{B}})$.
- If $(0 < l^{\mathcal{A}})$, and for all i from 0 to l:

$$m^{\mathcal{A}}(a_2).arr^{\mathcal{A}}(i) = null^{\mathcal{A}} \tag{7}$$

$$path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, i+1)) \subseteq path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, i)) \tag{8}$$

  Then $0 < l^{\mathcal{B}}$ (because 0 is never removed)

This concludes the proof. □

Theorem 1. *A* TSL *formula φ is satisfiable if and only if for some arrangement α, both $(\varphi^{PA} \wedge \alpha)$ and $(\varphi^{NC} \wedge \alpha)$ are satisfiable.*

Proof. The "⇒" direction follows immediately, since a model of φ contains a model of its subformulas φ^{PA} and φ^{NC}, and a model of φ^{PA} induces a satisfying order arrangement α.

For "⇐", let α be an order arrangement for which both $(\varphi^{\text{PA}} \wedge \alpha)$ and $(\varphi^{\text{NC}} \wedge \alpha)$ are satisfiable, and let $\mathcal{A}$ be a model of $(\varphi^{\text{NC}} \wedge \alpha)$ and $\mathcal{B}$ be a model of $(\varphi^{\text{PA}} \wedge \alpha)$. By Corollary 1, we assume that $\mathcal{A}$ is a gapless model. In particular, for all variables $l \in V_{\mathsf{level}}(\varphi)$, then $l^{\mathcal{A}} < \mathsf{K}$, where $\mathsf{K} = |V_{\mathsf{level}}(\varphi)|$, and for all cells $c \in \mathcal{A}_{\mathsf{cell}}$, with $c = (e, k, A, l)$, $l < \mathsf{K}$. Model $\mathcal{B}$ of $(\varphi^{\text{PA}} \wedge \alpha)$ assigns values to variables from $V_{\mathsf{level}}(\varphi)$, consistently with α. The obstacle is that the values for levels in $\mathcal{A}$ and in $\mathcal{B}$ may be different, so the models cannot be immediately merged. We will build a model $\mathcal{C}$ of φ using $\mathcal{A}$ and $\mathcal{B}$. Let K^{PA} be the largest value assigned by $\mathcal{B}$ to any variable from $V_{\mathsf{level}}(\varphi)$. We start by defining the following maps:

$$f : [\mathsf{K}] \to [\mathsf{K}^{\text{PA}}] \qquad\qquad f^* : [\mathsf{K}^{\text{PA}}] \to [\mathsf{K}]$$
$$l^{\mathcal{A}} \mapsto l^{\mathcal{B}} \qquad\qquad n \mapsto max\{k \in [\mathsf{K}] \mid f(k) \leq n\}$$

Essentially, f^* provides the level from $\mathcal{A}$ that will be used to fill the missing level in model $\mathcal{C}$. Some easy facts that follow from the choice of the definition of f and f^* are that, for every variable l in $V_{\mathsf{level}}(\varphi)$, $f^*(f(l^{\mathcal{A}})) = l^{\mathcal{A}}$. Also, every literal of the form $B = A\{l \leftarrow a\}$ satisfies that $f^*(l+1) = f^*(l) + 1$ because a sanitized formula φ contains a literal $l_{new} = l + 1$ for every such $B = A\{l \leftarrow a\}$.

We show now how to build a model $\mathcal{C}$ of φ. The only literals missing in φ^{NC} with respect to φ are literals of the form $l = q$ for constant level q. $\mathcal{C}$ agrees with $\mathcal{A}$ on sorts $\mathsf{addr}, \mathsf{elem}, \mathsf{ord}, \mathsf{path}, \mathsf{set}$. Also the domain $\mathcal{C}_{\mathsf{level}}$ is the naturals with order, and $\mathcal{C}_{\mathsf{cell}} = \mathcal{C}_{\mathsf{elem}} \times \mathcal{C}_{\mathsf{ord}} \times \mathcal{C}_{\mathsf{array}} \times \mathcal{C}_{\mathsf{level}}$ and $\mathcal{C}_{\mathsf{mem}} = \mathcal{C}_{\mathsf{cell}}^{\mathcal{C}_{\mathsf{addr}}}$. For level variables, we let $v^{\mathcal{C}} = v^{\mathcal{B}}$, where $v^{\mathcal{B}}$ is the interpretation of variable v in $\mathcal{B}$, the model of $(\varphi^{\text{PA}} \wedge \alpha)$. Note that $v^{\mathcal{C}} = v^{\mathcal{B}} = f(v^{\mathcal{A}})$. For arrays, we define $\mathcal{C}_{\mathsf{array}}$ to be the set of arrays of addresses indexed by naturals, and define the transformation $\beta : \mathcal{A}_{\mathsf{array}} \to \mathcal{C}_{\mathsf{array}}$ as follows: $\beta_{\mathsf{array}}(A)(i) = A(f^*(i))$.

Then, elements of sort cell $c : (e, k, A, l)$ are transformed into $\beta_{\mathsf{cell}}(c) = (e, k, \beta_{\mathsf{array}}(A), f(l))$. Variables of sort array A are interpreted as $A^{\mathcal{C}} = \beta_{\mathsf{array}}(A^{\mathcal{A}})$

and variables of sort cell as $\beta_{\mathsf{cell}}(c)$. Finally, heaps are transformed by returning the transformed cell: for $v \in V_{\mathsf{mem}}$, $v^{\mathcal{C}}(a) = \beta_{\mathsf{cell}}(v^{\mathcal{A}})(a)$. We only need to show that $\mathcal{C}$ is indeed a model of φ. Interestingly, all literals $l = q$ in $\mathcal{C}$ are immediately satisfied because $l^{\mathcal{C}} = l^{\mathcal{B}}$ and $q^{\mathcal{C}} = q^{\mathcal{B}}$, and the literal $(l = q)$ holds in the model $\mathcal{B}$ of φ^{PA}. The same holds for all literals in φ of the form $l_1 < l_2$, $l_1 = l_2 + 1$ and $l_1 \neq l_2$: these literals hold in $\mathcal{C}$ because they hold in $\mathcal{B}$. The following literals also hold in $\mathcal{C}$ because they hold in $\mathcal{A}$ and their subformulas either receive the same values in $\mathcal{C}$ than in $\mathcal{A}$ or the transformations are the same:

$e_1 \neq e_2$	$a_1 \neq a_2$	
$a = \mathit{null}$	$c = \mathit{error}$	$c = \mathit{rd}(m, a)$
$k_1 \neq k_2$	$k_1 \preceq k_2$	$m_2 = \mathit{upd}(m_1, a, c)$
$c = \mathit{mkcell}(e, k, A, l)$		
$s = \{a\}$	$s_1 = s_2 \cup s_3$	$s_1 = s_2 \setminus s_3$
$p_1 \neq p_2$	$p = [a]$	$p_1 = \mathit{rev}(p_2)$
$s = \mathit{path2set}(p)$	$\mathit{append}(p_1, p_2, p_3)$	$\neg\mathit{append}(p_1, p_2, p_3)$
		$\mathit{ordList}(m, p)$

Finally, observe that $(s = \mathit{addr2set}(m, a, l))$ and $(p = \mathit{getp}(m, a_1, a_2, l))$ hold in $\mathcal{C}$ whenever they hold in $\mathcal{A}$, as they follow directly from Lemma 2. The remaining literals are:

– $a = A[l]$: assume $a^{\mathcal{A}} = A^{\mathcal{A}}[l^{\mathcal{A}}]$. Then, in $\mathcal{C}$, $a^{\mathcal{C}} = a^{\mathcal{C}}$ and

$$A^{\mathcal{C}}[l^{\mathcal{C}}] = \beta(A^{\mathcal{A}})[l^{\mathcal{C}}] = A^{\mathcal{A}}(f^*(l^{\mathcal{C}})) = A^{\mathcal{A}}(f^*(f(l^{\mathcal{A}}))) = A^{\mathcal{A}}(l^{\mathcal{A}}) = a^{\mathcal{A}} = a^{\mathcal{C}}.$$

– $B = A\{l \leftarrow a\}$: We distinguish two cases. First, let $n = l^{\mathcal{C}}$. Then,

$$A^{\mathcal{C}}\{l^{\mathcal{C}} \leftarrow a\}(n) = A^{\mathcal{C}}\{l^{\mathcal{C}} \leftarrow a\}(l^{\mathcal{C}}) = a, \text{ and}$$
$$B^{\mathcal{C}}(n) = B^{\mathcal{C}}(l^{\mathcal{C}}) = B^{\mathcal{A}}(f^*(l^{\mathcal{C}})) = B^{\mathcal{A}}(f^*(f(l^{\mathcal{A}}))) = B^{\mathcal{A}}(l^{\mathcal{A}}) = a.$$

The second case is $n \neq l^{\mathcal{C}}$. Then $(A^{\mathcal{C}}\{l^{\mathcal{C}} \leftarrow a\})(n) = A^{\mathcal{C}}(n) = A^{\mathcal{A}}(f^*(n))$, and $B^{\mathcal{C}}(n) = B^{\mathcal{A}}(f^*(n))$. Now, $f^*(n) \neq l^{\mathcal{A}}$. To show this we consider the two cases for $n \neq l^{\mathcal{C}}$:

- If $n < l^{\mathcal{C}}$ then, since $f^*(n) = \max\{k \in [\mathsf{K}] | f(k) \leq n\}$ by definition, $f(l^{\mathcal{A}}) = l^{\mathcal{C}} > n$ and $f^*(n) < l^{\mathcal{A}}$ which implies $f^*(n) \neq l^{\mathcal{A}}$.
- If $n > l^{\mathcal{C}}$ then $n \geq l^{\mathcal{C}} + 1$. As reasoned above there is a different literal $l_{new} = l + 1$ for which $f^*(n) \geq f^*(l^{\mathcal{C}}_{new}) > f^*(l^{\mathcal{C}}) = l^{\mathcal{A}}$

Since in both cases $f^*(n) \neq l^{\mathcal{A}}$, then

$$B^{\mathcal{C}}(n) = B^{\mathcal{A}}(f^*(n)) = A^{\mathcal{A}}(f^*(n)) = A^{\mathcal{A}}\{l^{\mathcal{A}} \leftarrow a\}(f^*(n)) = A^{\mathcal{C}}\{l^{\mathcal{C}} \leftarrow a\}(n)$$

Essentially, the choice to introduce a variable $l_{new} = l + 1$ restricts the replication of identical levels to only the level l in $B = A\{l \leftarrow a\}$. All higher and lower levels are replicas of levels different than l (where A and B agree as in model $\mathcal{A}$).

– $s = \mathit{addr2set}(m, a, l)$ and $p = \mathit{getp}(m, a_1, a_2, l)$: it is easy to show by induction on the length of paths that, for all $l^{\mathcal{A}}$:

$$(m^{\mathcal{A}}, a^{\mathcal{A}}, b^{\mathcal{A}}, l^{\mathcal{A}}, p^{\mathcal{A}}) \in \mathit{reach}^{\mathcal{A}} \quad \mathit{iff} \quad (m^{\mathcal{C}}, a^{\mathcal{C}}, b^{\mathcal{C}}, l^{\mathcal{C}}, p^{\mathcal{C}}) \in \mathit{reach}^{\mathcal{C}} \tag{9}$$

It follows that $s^{\mathcal{A}} = addr2set(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}})$ implies $s^{\mathcal{C}} = addr2set(m^{\mathcal{C}}, a^{\mathcal{C}}, l^{\mathcal{C}})$. Also $p^{\mathcal{A}} = getp(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, l^{\mathcal{A}})$ implies that $p^{\mathcal{C}} = getp(m^{\mathcal{C}}, a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, l^{\mathcal{C}})$. Essentially, since level $l^{\mathcal{C}}$ in $\mathcal{C}$ is a replica of level $l^{\mathcal{A}}$ in $\mathcal{A}$, the transitive closure of following pointers is the same paths (for *getp*) and the same also the same sets (for *addr2set*).

- $skiplist(m, r, l, a_1, a_2)$. We assume $skiplist(m^{\mathcal{A}}, r^{\mathcal{A}}, l^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}})$. This implies all of the following in $\mathcal{A}$:
  - $ordList^{\mathcal{A}}(m^{\mathcal{A}}, getp^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. Let p be such that $p = getp^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. As a consequence of (9) $p = getp^{\mathcal{C}}(a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, 0))$, and then
  $$ordList^{\mathcal{A}}(m^{\mathcal{A}}, getp^{\mathcal{A}}(a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0)) \quad \text{implies} \quad ordList^{\mathcal{C}}(m^{\mathcal{C}}, getp^{\mathcal{C}}(a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, 0)).$$
  - $r^{\mathcal{A}} = path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0))$. Because $getp^{\mathcal{C}}(m^{\mathcal{C}}, a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, 0) = getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, 0)$, once more $r^{\mathcal{C}} = path2set^{\mathcal{C}}(getp^{\mathcal{C}}(m^{\mathcal{C}}, a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, 0))$.
  - $0 \leq l^{\mathcal{A}}$, which implies $0 \leq l^{\mathcal{C}}$.
  - $\forall a \in r^{\mathcal{A}} \,.\, m^{\mathcal{A}}(a^{\mathcal{A}}).max^{\mathcal{A}} \leq l^{\mathcal{A}}$. Since $r^{\mathcal{C}} = r^{\mathcal{A}}$ and $m^{\mathcal{C}}(a) = \gamma(m^{\mathcal{A}}(a))$ it is enough to consider two cases. First, $m^{\mathcal{A}}(a).max^{\mathcal{A}} = l^{\mathcal{A}}$, in which case $m^{\mathcal{C}}(a).max^{\mathcal{C}} = l^{\mathcal{C}}$. Second $m^{\mathcal{A}}(a).max^{\mathcal{A}} < l^{\mathcal{A}}$, in which case $m^{\mathcal{A}}(a).max^{\mathcal{A}} \leq l^{\mathcal{A}}$.
  - If $(0 = l^{\mathcal{A}})$ then $(0 = l^{\mathcal{C}})$.
  - If $(0 < l^{\mathcal{A}})$, and for all i from 0 to $l^{\mathcal{A}}$:
  $$\begin{aligned} & m^{\mathcal{A}}(a_2).arr^{\mathcal{A}}(i) = null^{\mathcal{A}} \\ & path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, i+1)) \subseteq \\ & path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, i)) \end{aligned}$$

  Then $0 < l^{\mathcal{C}}$. Consider an arbitrary i between 0 and $l^{\mathcal{C}}$. It follows that $f^*(i) \leq f^*(l^{\mathcal{C}})$ so $f^*(i) \leq l^{\mathcal{A}}$ and then
  $$\begin{aligned} & m^{\mathcal{C}}(a_2).arr^{\mathcal{C}}(i) = m^{\mathcal{C}}(a_2).arr^{\mathcal{A}}(f^*(i)) = null^{\mathcal{A}} = null^{\mathcal{C}} \\ & path2set^{\mathcal{C}}(getp^{\mathcal{C}}(m^{\mathcal{C}}, a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, i+1)) = \\ & path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, f^*(i+1))) \subseteq \\ & path2set^{\mathcal{A}}(getp^{\mathcal{A}}(m^{\mathcal{A}}, a_1^{\mathcal{A}}, a_2^{\mathcal{A}}, f^*(i))) = \\ & path2set^{\mathcal{C}}(getp^{\mathcal{C}}(m^{\mathcal{C}}, a_1^{\mathcal{C}}, a_2^{\mathcal{C}}, i)) \end{aligned}$$

This concludes the proof. □

Lemma 5. *Let ψ be a sanitized* TSL *formula with no constants. Then, ψ is satisfiable if and only if $\ulcorner\psi\urcorner$ is also satisfiable.*

Proof. Directly from Lemmas 6 and 7 below, which prove each direction separately. □

Lemma 6. *Let φ be a normalized set of* TSL *literals with no constants. Then, if φ is satisfiable then $\ulcorner\varphi\urcorner$ is also satisfiable.*

Proof. Assume φ is satisfiable, which implies (by Corollary 1) that φ has a gapless model $\mathcal{A}$. This model $\mathcal{A}$ satisfies that for every natural i from 0 to $\mathsf{K}-1$ there is a level $l \in V_{\mathsf{level}}(\varphi)$ with $l^{\mathcal{A}} = i$.

Building a Model $\mathcal{B}$ We now construct a model $\mathcal{B}$ of $\ulcorner\varphi\urcorner$. For the domains:

$$\mathcal{B}_{\mathsf{addr}} = \mathcal{A}_{\mathsf{addr}} \qquad \mathcal{B}_{\mathsf{elem}} = \mathcal{A}_{\mathsf{elem}} \qquad \mathcal{B}_{\mathsf{ord}} = \mathcal{A}_{\mathsf{ord}} \qquad \mathcal{B}_{\mathsf{path}} = \mathcal{A}_{\mathsf{path}} \qquad \mathcal{B}_{\mathsf{set}} = \mathcal{A}_{\mathsf{set}}$$

and

$$\mathcal{B}_{\mathsf{level}} = [\mathsf{K}] \qquad \mathcal{B}_{\mathsf{cell}} = \mathcal{B}_{\mathsf{elem}} \times \mathcal{B}_{\mathsf{ord}} \times \mathcal{B}_{\mathsf{addr}}^{\mathsf{K}} \qquad \mathcal{B}_{\mathsf{mem}} = \mathcal{B}_{\mathsf{cell}}^{\mathcal{B}_{\mathsf{addr}}}$$

For the variables, we let $v^{\mathcal{B}} = v^{\mathcal{A}}$ for sorts addr, elem, ord, path and set. For level, we assign $l^{\mathcal{B}} = l^{\mathcal{A}}$, which is guaranteed to be within 0 and $\mathsf{K} - 1$. For cell, let $c = (e, k, A, l)$ be an element of $\mathcal{A}\mathsf{cell}$. The following function maps c into an element of $\mathcal{B}\mathsf{cell}$:

$$\alpha(e, k, A, l) = (e, k, A(0), \ldots, A(\mathsf{K} - 1))$$

Essentially, cells only record information of relevant levels, which are those levels for which there is a level variable; all upper levels are ignored. Every variable v of sort cell is interpreted as $v^{\mathcal{B}} = \alpha(v^{\mathcal{A}})$. Finally, a variable v of sort mem is interpreted as a function that maps an element a of $\mathcal{B}\mathsf{addr}$ into $\alpha(v^{\mathcal{A}}(a))$, essentially mapping addresses to transformed cells. Finally, for all arrays A in the formula φ, we assign $v^{\mathcal{B}}_{A[i]} = A^{\mathcal{A}}(i)$.

Checking the Model $\mathcal{B}$ We are ready to show, by case analysis on the literals of the original formula φ, that $\mathcal{B}$ is indeed a model of $\ulcorner\varphi\urcorner$. The following literals hold in $\mathcal{B}$, directly from the choice of assignments in $\mathcal{B}$ because the corresponding literals hold in $\mathcal{A}$:

$e_1 \neq e_2$	$a_1 \neq a_2$	$l_1 \neq l_2$
$a = \mathit{null}$	$c = \mathit{error}$	$c = \mathit{rd}(m, a)$
$k_1 \neq k_2$	$k_1 \preceq k_2$	$m_2 = \mathit{upd}(m_1, a, c)$
	$l_1 < l_2$	$l = q$
$s = \{a\}$	$s_1 = s_2 \cup s_3$	$s_1 = s_2 \setminus s_3$
$p_1 \neq p_2$	$p = [a]$	$p_1 = \mathit{rev}(p_2)$
$s = \mathit{path2set}(p)$	$\mathit{append}(p_1, p_2, p_3)$	$\neg\mathit{append}(p_1, p_2, p_3)$
		$\mathit{ordList}(m, p)$

The remaining literals are:

- $c = \mathit{mkcell}(e, k, A, l)$: Clearly the data and key fields of $c^{\mathcal{B}}$ and the translation of $\mathit{mkcell}^{\mathcal{B}}(e, k, \ldots)$ coincide. Similarly, by the α map for elements of $\mathcal{B}\mathsf{cell}$, the array entries coincide with the values of the fresh variables $v_{A[i]}$. Hence, $c = \mathit{mkcell}(e, k, v_{A[0]}, \ldots, v_{A[K-1]})$ holds in $\mathcal{B}$.
- $a = A[l]$: our choice of $v^{\mathcal{B}}_{A[i]}$ makes

$$v^{\mathcal{B}}_{A[i]} = A^{\mathcal{A}}(l^{\mathcal{B}}) = A^{\mathcal{A}}(l^{\mathcal{A}}) = a^{\mathcal{A}} = a^{\mathcal{B}}$$

so the clause generated from $a = A[l]$ in $\ulcorner\varphi\urcorner$ holds in $\mathcal{B}$.

- $B = A\{l \leftarrow a\}$: In this case, for $j = l^{\mathcal{A}} = l^{\mathcal{B}}$, $v^{\mathcal{B}}_{B[j]} = B^{\mathcal{A}}(l^{\mathcal{A}}) = a^{\mathcal{A}} = a^{\mathcal{B}}$. Moreover, for all other indices i:
$$v^{\mathcal{B}}_{B[i]} = B^{\mathcal{A}}(i) = A^{\mathcal{A}}(i) = v^{\mathcal{B}}_{A[i]}$$
so the clause (1) generated from $B = A\{l \leftarrow a\}$ in $\ulcorner\varphi\urcorner$ holds in $\mathcal{B}$.
- $s = \mathit{addr2set}(m, a, l)$: it is easy to show by induction on the length of paths that, for all $l^{\mathcal{A}}$:
$$(m^{\mathcal{A}}, a^{\mathcal{A}}, b^{\mathcal{A}}, l^{\mathcal{A}}, p^{\mathcal{A}}) \in \mathit{reach}^{\mathcal{A}} \quad \mathit{iff} \quad (m^{\mathcal{B}}, a^{\mathcal{B}}, b^{\mathcal{B}}, l^{\mathcal{B}}, p^{\mathcal{B}}) \in \mathit{reach}^{\mathcal{B}} \tag{10}$$
It follows that $s^{\mathcal{A}} = \mathit{addr2set}(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}})$ implies $s^{\mathcal{B}} = \mathit{addr2set}(m^{\mathcal{B}}, a^{\mathcal{B}}, l^{\mathcal{B}})$.
- $p = \mathit{getp}(m, a_1, a_2, l)$: Fact (10) also implies immediately that if literal $p = \mathit{getp}(m, a_1, a_2, l)$ holds in $\mathcal{A}$ then $p = \mathit{getp}(m, a_1, a_2, l)$ holds in $\mathcal{B}$.
- $\mathit{skiplist}(m, s, a_1, a_2)$: Following (2) the four disjuncts (1) the lowest level is ordered, (2) the region contains exactly all low level, (3) the centinel cell has null successors, and (4) each level is a subset of the lower level, hold in $\mathcal{B}$, because they corresponding disjunct holds in $\mathcal{A}$.

This shows that $\mathcal{B}$ is a model of $\ulcorner\varphi\urcorner$ and therefore $\ulcorner\varphi\urcorner$ is satisfiable. □

Lemma 7. *Let φ be a normalized set of TSL literals with no constants. If $\ulcorner\varphi\urcorner$ is satisfiable, then φ is also satisfiable.*

Proof. We start from a $\mathsf{TSL_K}$ model $\mathcal{B}$ of $\ulcorner\varphi\urcorner$ and construct a model $\mathcal{A}$ of φ.

Building a Model $\mathcal{A}$ We now proceed to show that φ is satisfiable by building a model $\mathcal{A}$. For the domains, we let:

$$\mathcal{A}_{\mathsf{addr}} = \mathcal{B}_{\mathsf{addr}} \qquad \mathcal{A}_{\mathsf{elem}} = \mathcal{B}_{\mathsf{elem}} \qquad \mathcal{A}_{\mathsf{ord}} = \mathcal{B}_{\mathsf{ord}} \qquad \mathcal{A}_{\mathsf{path}} = \mathcal{B}_{\mathsf{path}} \qquad \mathcal{A}_{\mathsf{set}} = \mathcal{B}_{\mathsf{set}}.$$

Also, $\mathcal{A}_{\mathsf{level}}$ is the naturals with order, and

$$\mathcal{A}_{\mathsf{cell}} = \mathcal{A}_{\mathsf{elem}} \times \mathcal{A}_{\mathsf{ord}} \times \mathcal{A}_{\mathsf{array}} \times \mathcal{A}_{\mathsf{level}} \qquad\qquad \mathcal{A}_{\mathsf{mem}} = \mathcal{A}_{\mathsf{cell}}^{\mathcal{A}_{\mathsf{addr}}}.$$

For the variables, we let $v^{\mathcal{A}} = v^{\mathcal{B}}$ for sorts addr, elem, ord, path and set. For level, we also assign $l^{\mathcal{A}} = l^{\mathcal{B}}$. For cell, let $c = (e, k, a_0, \ldots, a_{\mathsf{K}-1})$ be an element of $\mathcal{B}_{\mathsf{cell}}$. Then the following function β maps c into an element of $\mathcal{A}_{\mathsf{cell}}$:

$$\beta(c : (e, k, a_0, \ldots, a_{\mathsf{K}-1})) = (e, k, A, l) \qquad \text{where} \tag{11}$$

$$l = \mathsf{K}$$
$$A(i) = \begin{cases} a_i & \text{if } 0 \le i < l \\ \mathit{null} & \text{if } i \ge l \end{cases}$$

Every variable v of sort cell is interpreted as $v^{\mathcal{A}} = \beta(v^{\mathcal{B}})$. Finally, a variable v of sort mem is interpreted as a function that maps an element a of $\mathcal{A}_{\mathsf{addr}}$ into $\beta(v^{\mathcal{B}}(a))$, mapping addresses to transformed cells.

Finally, for all arrays variables A in the original formula φ, we assign:

$$A^{\mathcal{A}}(i) = \begin{cases} v^{\mathcal{B}}_{A[i]} & \text{if } i < \mathsf{K} \\ \mathit{null} & \text{otherwise} \end{cases} \tag{12}$$

Checking the Model $\mathcal{A}$ We are ready to show, by cases on the literals of the original formula φ that $\mathcal{A}$ is indeed a model of φ. The following literals hold in $\mathcal{A}$ because the corresponding literals hold in $\mathcal{B}$:

$e_1 \neq e_2$	$a_1 \neq a_2$	$l_1 \neq l_2$
$a = null$	$c = error$	$c = rd(m, a)$
$k_1 \neq k_2$	$k_1 \preceq k_2$	$m_2 = upd(m_1, a, c)$
	$l_1 < l_2$	$l = q$
$s = \{a\}$	$s_1 = s_2 \cup s_3$	$s_1 = s_2 \setminus s_3$
$p_1 \neq p_2$	$p = [a]$	$p_1 = rev(p_2)$
$s = path2set(p)$	$append(p_1, p_2, p_3)$	$\neg append(p_1, p_2, p_3)$
		$ordList(m, p)$

The remaining literals are:

- $c = mkcell(e, k, A, l)$: Clearly the data and key fields of $c^{\mathcal{A}}$ and the translation of $mkcell^{\mathcal{A}}(e, k, \ldots)$ given by (11) coincide. By the choice of array variables $A^{\mathcal{A}}(i) = v^{\mathcal{B}}_{A[i]} = a_i$, so A and the array part of c coincide at all positions. For $l^{\mathcal{A}}$ we choose K for all cells.
- $a = A[l]$: holds since
$$a^{\mathcal{A}} = a^{\mathcal{B}} = v^{\mathcal{B}}_{A[l^{\mathcal{B}}]} = A^{\mathcal{A}}(l^{\mathcal{B}}) = A^{\mathcal{A}}(l^{\mathcal{A}}).$$
- $B = A\{l \leftarrow a\}$: We have that the translation of $B = A\{l \leftarrow a\}$ for $\ulcorner\varphi\urcorner$ given by (1) holds in $\mathcal{B}$. Consider an arbitrary level $m < \mathsf{K}$. If $m = l^{\mathcal{B}} = l^{\mathcal{A}}$ then $a = v_{B[m]} = B^{\mathcal{A}}(m)$. If $m \neq l^{\mathcal{B}}$ then $v_{B[m]} = v_{A[m]}$ and hence $B^{\mathcal{A}}(m) = v_{B[m]} = v_{A[m]} = A^{\mathcal{A}}(m)$.
- $A = B$: the clause (12) generated from $A = B$ in $\ulcorner\varphi\urcorner$ holds in $\mathcal{B}$, by assumption. For an arbitrary j from $[\mathsf{K}]$:
$$A^{\mathcal{A}}(j) = v^{\mathcal{B}}_{A[j]} = v^{\mathcal{B}}_{B[j]} = B^{\mathcal{A}}(j)$$
Moreover, for $j \geq \mathsf{K}$, then $A^{\mathcal{A}}(j) = null = B^{\mathcal{A}}(j)$ and consequently $A^{\mathcal{A}} = B^{\mathcal{A}}$ as desired.
- $s = addr2set(m, a, l)$: it is easy to show by induction on the length of paths that, for all $l^{\mathcal{A}}$:
$$(m^{\mathcal{A}}, a^{\mathcal{A}}, b^{\mathcal{A}}, l^{\mathcal{A}}, p^{\mathcal{A}}) \in reach^{\mathcal{A}} \quad \textit{iff} \quad (m^{\mathcal{B}}, a^{\mathcal{B}}, b^{\mathcal{B}}, l^{\mathcal{B}}, p^{\mathcal{B}}) \in reach^{\mathcal{B}} \qquad (13)$$
It follows that $s^{\mathcal{A}} = addr2set(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}})$ implies $s^{\mathcal{A}} = addr2set(m^{\mathcal{A}}, a^{\mathcal{A}}, l^{\mathcal{A}})$.
- $p = getp(m, a_1, a_2, l)$: Fact (13) also implies immediately that if literal $p = getp(m, a_1, a_2, l)$ holds in $\mathcal{A}$ then $p = getp(m, a_1, a_2, f(l))$ holds in $\mathcal{B}$.
- $skiplist(m, s, a_1, a_2)$: Following (2) the four disjuncts (1) the lowest level is ordered, (2) the region contains exactly all low addresses in the lowest level, (3) the centinel cell has null successors, and (4) each level is a subset of the lower level, hold in $\mathcal{A}$, because they corresponding disjunct holds in $\mathcal{B}$.

This shows that $\mathcal{A}$ is a model of φ and therefore φ is satisfiable. $\square$