Review: Formal verification of skiplists with arbitrary many levels

Tanguy Rocher
tanguy.rocher@epfl.ch

Hugo Lepeytre
hugo.lepeytre@epfl.ch

Paul Juillard
paul.juillard@epfl.ch

November 13, 2021

1 Introduction

The paper [2] presents a method for formal verification of imperative skiplists, which is a data structure composed of several ordered singly-linked lists arranged as levels. They are of importance because they allow for an easier implementation of sets than balanced trees while having comparable performance. Until then, verification of skiplists was limited to a bounded number of levels, mainly because of the complexity induced by the memory sharing between levels. In the paper, a new theory of skiplists is developed allowing formal verification of skiplists with arbitrarily many levels as well as arbitrary length. Furthermore, the algorithm terminates quickly no matter the number of levels in the list, while the previous verification algorithms did not scale beyond 3 levels. The paper also contains empirical measuring of performance on the verification of two source code implementations of skiplists, one part of the industrial open source project KDE and the other developed internally.

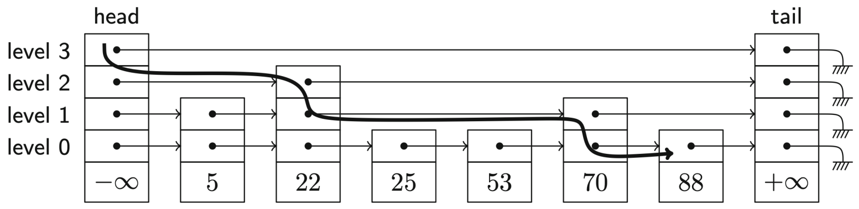

Figure 1: A skiplist with 4 levels and the traversal searching 88 [2]

2 Preliminaries

Necessary background concepts from the paper will be discussed in turn : skiplists, Presburger Arithmetic, Expressing the theory of skiplists through First-Order Logic.

2.1 Skiplists

A skiplist is a data structure consisting of singly-linked ordered lists organised as levels : The list at level 0 contains all elements in the set while every level $i > 0$ contains a subset of the elements at level $i - 1$. Every element at level i links to the next element on level i as well as to itself on level $i - 1$. With the structure in mind, 3 main operations are provided : insert, search and remove. As an example, the search operation is shown in Figure 2.1.
Insert is performed by first deciding a random level i for the new element (can be higher than the current number of levels. Then if the element was already present in th set, we do nothing, otherwise it is simply inserted in all linked lists at levels $\leq i$.
For searching value v, we start at the highest level (i) and find the highest element v_i such that $v_i \leq v$. Then we go down by a level, starting this time at element v_i and finding v_{i-1}. We repeat the process until we get to level 0. We then check if $v_0 = v$. If it is not, then the set does not contain v.
Removing is performed by finding the highest level i containing v, and then removing v for every linked list in levels $\leq i$.
The complexity of those 3 operations is probabilistic since every element is inserted at a random level, but their expected complexity is $\mathrm{O}(\log n)$.

2.2 Presburger Arthmetic

The proof and verification procedure also make use of Presburger arithmetic, which is the first-order logic description of natural numbers with addition. Its properties are consistency, completeness and most importantly here, decidability. This last property is useful because descriptions of skiplists in the language developed in the paper can be reduced to a mix of Presburger arithmetic formulas and TSL_k formulas (which are described in the next section). Both of them are proved to be decidable, making TSL itself decidable.

2.3 TSL$_k$

TSL$_k$ is a theory of concurrent skiplists of height k, developed by the same authors in a previous paper [1]. In particular it allows representation of pointer data structures, memory usage and concurrency locks, and it is very similar to the TSL theory which will be described below. It has also been proven to be decidable in the same paper in which it was described.

3 TSL

3.1 TSL theory

We have seen TSL_k in the preliminaries. This paper's main contribution is the construction of the more general TSL theory, i.e. abstracting away from the hyper-parameter k. Previously, practical limitations constrained $k < 4$. Further than a higher-level object, the generalized reasoning additionally allows to lift the aforementioned bound in practice.

Formally, the TSL signature is constructed as: is TSL a first order Theory?

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

Where signatures for *level*, *set* and *ord*(ering) are rather self-explanatory and constructed from base types. Noteworthy, *ord* accommodates for the head and tail elements $-\inf$ and $+\inf$. Elements of the skiplist are called *cells* and Σ_{cell} is element operations e.g. construction or inspection. Skiplist implementations keep pointers to these cells in arrays. Σ_{array} builds on previous works [add citation] to provide a signature for the following two basic operations: $A[i]$ returns the pointer to the element at i in array A, while $A\{i \leftarrow p\}$ updates the array. Pointer and memory addressing logic is done through the *mem* signatures.

not so much. just enumerate symbols, it will be clearer and not longer.

This leaves Σ_{reach} and Σ_{bridge} for which we reproduce the explicit definitions [2], providing insight.

Σ_{reach}

- ϵ : path
- $[\_]$: addr $\rightarrow$ path
- *append* : path $\times$ path $\times$ path why not path $\times$ path $\rightarrow$ path?
- *reach* : mem $\times$ addr $\times$ addr $\times$ level $\times$ path

, function, not relation?

ϵ and $[\_]$ denote the empty and singleton paths. *append* concatenates the second path to the first yielding the third. Finally, $reach = (mem, a_{init}, a_{end}, level, p)$ represents the fact that there is a path in *mem* from a_{init} to a_{end} along path $p = [a_{init}, a_2, ..., a_{end}]$ at level *level*.

Σ_{bridge}

- $path2set$: path $\rightarrow$ set
- $addr2set$: mem $\times$ addr $\times$ level $\rightarrow$ set
- $getp$: mem $\times$ addr $\times$ addr $\times$ level $\rightarrow$ path
- $ordList$: mem $\times$ path
- $skiplist$: mem $\times$ set $\times$ level $\times$ addr $\times$ addr

This signature incorporates a few utility functions of little interest, but for the *skiplist* predicate. It formally constructs from all of the above signatures the condition for an object to be a skiplist as the conjunction of the following: the elements form an ordered set, with a maximum level, and underlying levels contain all overlying ones, as seen in the preliminaries.

you're mixing signature and) semantics/ interpretation

After a lengthy two pages of signature definitions and Presburger arithmetic tables, we have our TSL theory. It is left to show how to use all this nice work and the additional benefit from TSL_k. The first is theoretical: the satisfiability problem on quantifier-free TSL formulas can be shown to be decidable. We reproduce a sketch of the formal proof in the next section. The second is practical: the authors use TSL to formally verify implementations of skiplists and break the previous limitations of verification for shallow skiplists with few levels. We also briefly discuss these results before concluding.

3.2 Decidability of satisfiability of QF TSL

Before actually diving into (sketched) technicalities of the proof, recall:

- quantifier-free Presburger arithmetic formula satisfiability is decidable
- quantifier-free TSL_k formula satisfiability is decidable

a procedure isn't decidable terminating?

The proof constructs a procedure for satisfiability checking that is decidable. The plan is to reduce the TSL formula to an equi-satisfiable combination of finitely many formulas in plain Presburger arithmetic and/or TSL_k theory.

The first step of the procedure is to sanitize the formula to a given restrained set of literals using basic conversions and makes sure there is (i.e. adds if necessary) a 'top-level' l_T that is not referred to in any assignment $A\{l \leftarrow p\}$. Further, if there are gaps in level references, for example level 1 and 3 are cited but never level 2, then an equivalent gap-less expression can be constructed by rewriting the level values.

Could be slightly more detailed

Next, the original formula ϕ is split in phi^{PA}, containing all literals from ϕ in the theory of Presburger arithmetic, and phi^{NC} which contains all literals from ϕ except those involving constants. As both sets of literals span all the literals from ϕ, we have that ϕ is equivalent to $\phi^{NC} \wedge \phi^{PA}$.

you mean literals in ϕ^{PA}?

From here, phi^{PA} satisfiability decidability follows from Presburger arithmetic properties. It is left to map phi^{NC} to an equi-satisfiable formula in TSL_k for a well-chosen k, the top-level of the reduced, gap-less expression. Knowing the maximum k allows to loop and translate all arbitrary-sized structures and expressions from TSL into corresponding, finitely many, TLS_k expressions.

Checking satisfiability for phi^{PA} and phi^{NC} is equi-satisfiable and decidable, which concludes the proof.

3.3 Formal Verification of implementations

The paper now show two empirical verification of the theory explained in last section. Two kind of properties were proved :

- The memory shape preservation of the structure (described as the predicate `skipList` in the last section)
- The functional Correctness of the skiplist, described as follow :
  - Searching for an element should not modify the skiplist. Additionally, the element is found if and only if it was in the skipList
  - Inserting an element should indeed add it to the skiplist
  - Remove an element should effectively remove it from the skiplist

To prove the first one however, the authors had to introduce auxiliary invariants. Their results show that TLS is a practical procedure to verify implementations with a variable number of levels as the running times were adequate for both implementations.

4 Conclusion

As we have seen, the paper has developed a first-order logic theory of skiplists and used it to come up with a fast and scalable way to verify skiplists. It looks very convincing, its only weak points being the fact that it is not completely automated, meaning some user input is needed to advance the verification process, especially when invariants are concerned. But even this is mentioned in the further work section, where it is suggested as a direction for future improvement alongside the verification of liveness properties of concurrent implementations.

For our purposes, this paper gives a good overview of skiplists and ideas for a verification method, but since we plan to give a functional implementation of skiplists, some of the ideas developed to manage verification of memory use will not be relevant and might have to be replaced with something more fitting of functional implementations.

References

[1] Alejandro Sánchez and César Sánchez. A theory of skiplists with applications to the verification of concurrent datatypes. In Holzmann G.J. Joshi R. Bobaru M., Havelund K., editor, *Automated Technology for Verification and Analysis*, pages 343–358, Berlin, Heidelberg, 2011. NASA Formal Methods.

[2] Alejandro Sánchez and César Sánchez. Formal verification of skiplists with arbitrary many levels. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 314–329, Cham, 2014. Springer International Publishing.