BmcMT: Bounded Model Checking of TLA + Specifications with SMT

Igor Konnov, Jure Kukovec, Thanh Tran

BmcMT: Bounded Model Checking of TLA$^+$ Specifications with SMT

Igor Konnov[1], Jure Kukovec[2], and Thanh Hai Tran[2]

[1]University of Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
Email: firstname.lastname@inria.fr
[2] TU Wien (Vienna University of Technology), Vienna, Austria *
Email: {jkukovec,tran}@forsyte.at

April 15, 2018

Abstract

We present the development version of BmcMT—a symbolic model checker for TLA$^+$. It finds, whether a TLA$^+$ specification satisfies an invariant candidate by checking satisfiability of an SMT formula that encodes: (1) an execution of bounded length, and (2) preservation of the invariant candidate in every state of the execution. Our tool is still in the experimental phase, due to a number of challenges posed by TLA$^+$ semantics to SMT solvers. We will discuss these challenges and our current approach to them in the talk. Our preliminary experiments show that BmcMT scales better than the standard TLA$^+$ model checker TLC for large parameter values, e.g., when a TLA$^+$ specification models a system of 10 processes, though TLC is more efficient for tiny parameters, e.g., when the system has 3 processes.

We believe that early feedback from the TLA$^+$ community will help us to focus on the most important language features and improve our tool.

1 Overview

Our work is motivated by the success story of software model checkers that use SAT and SMT solvers in the back-end, as well as by the work of Stephan Merz and Hernán Vanzetto on automating TLA$^+$ proofs with SMT [3]. In contrast to automated theorem provers, our goal is to develop a tool that, in principle, should work on all TLA$^+$ specifications that are accepted by TLC. Hence, to avoid quantifiers in SMT formulas, we use the following pragmatic assumptions, similar to the assumptions of TLC:

- **Finite states.** As in TLC, all sets, function domains, and function co-domains constructed during an execution of a TLA$^+$ specification should be finite.
- **Fixed and finite constants.** The user initializes all the specification parameters. Again, the parameter values should be finite.

*This research supported in part by the Vienna Science and Technology Fund (WWTF) through project APALACHE (ICT15-103).

- **Restrictions on the formula structure.** Under certain circumstances, we interpret an equality $x' = e$ as an assignment of the e's value to the variable x. Likewise, a membership test $x' \in S$ is sometimes interpreted as a non-deterministic assignment of values from the set S to the variable x. Finding such assignments symbolically, that is, without evaluating the specification, happened to be a non-trivial problem. We reported on a solution to this problem in [2].

In order to encode TLA$^+$ values in SMT, we are using a type system similar to [3]. However, our tool is running type inference on-the-fly: It finds the types for the initial states based on the user input; then, given the types of the values at the ith state, it finds the types of the values computed at the $(i+1)$th state. Due to this approach, type inference BmcMT is fairly simple.

To translate action-level TLA$^+$ expressions to SMT, we define operational semantics over finite TLA$^+$ values. To this end, our tool computes a static layout of the data structures that may be created during an execution of a TLA$^+$ specification. This static layout overapproximates the data structures that are computed in an actual execution, whereas the precise shape of the data structures is defined by constraints within the SMT solver. We use the static layout to define operations such as set equality, which otherwise would require us to write axioms similar to [3] and thus introduce quantified formulas. Similarly, quantified TLA$^+$ expressions $\forall x \in S.\ P$ and $\exists x \in S.\ P$ are expanded to finite disjunctions and conjunctions over potential elements of S. By doing so, we manage to stay within the theory of `QF-UFLIA`, that is, quantifier-free linear integer arithmetics with uninterpreted functions.

So far, we have defined and implemented semantics of the following TLA$^+$ features:

- Logical operators, except CHOOSE and unbounded quantifiers $\forall x.\ e$ and $\exists x.\ e$;
- Integer operators, including the range operator $a..b$ for constant a and b;
- Set operators over finite sets: $\{e_1, \ldots, e_k\}$, $\{x \in S : P\}$, $\{e : x \in S\}$, $x \in S$, $X \subset Y$, $X = Y$, $X \cup Y$, $X \cap Y$, SUBSET S, and UNION S.
- Function operators over functions with finite domains and co-domains: $[x \in S \mapsto e]$, $f[e]$, $[f$ EXCEPT $![e_1] = e_2]$, $[S \to T]$, DOMAIN f, and equality.
- Operators over tuples and records with constant domains, as well as IF-THEN-ELSE.

We believe that most of the other TLA$^+$ operators can be encoded in SMT using our approach. However, the following operators are particularly challenging: set cardinality and sequence operators with non-constant arguments like $x..y$. Interestingly, these operators are not a problem for TLC, as it computes the exact variable values by state enumeration.

The development version of the tool is available at [1].

References

[1] I. Konnov, J. Kukovec, and T. H. Tran. APALACHE BmcMT (development version). URL: `https://github.com/konnov/apalache/tree/unstable`. Accessed: Apr, 2018.

[2] J. Kukovec, T. H. Tran, and I. Konnov. Extracting symbolic transitions from TLA$^+$ specifications. In *ABZ 2018, 6th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z*, 2018. (To appear). URL: `http://forsyte.at/wp-content/uploads/abz2018_full.pdf`.

[3] S. Merz and H. Vanzetto. Encoding TLA$^+$ into many-sorted first-order logic. In *ABZ*, pages 54–69, 2016.