Proofs and Proof Certification in the TLA$^+$ Proof System

Stephan Merz

Inria Nancy Grand-Est & LORIA, Villers-lès-Nancy, France

Abstract

TLA$^+$ is a specification language originally designed for specifying concurrent and distributed systems and their properties. It is based on Zermelo-Fraenkel set theory for modeling data structures and on the linear-time temporal logic TLA for specifying system executions and their properties. The TLA$^+$ proof system (TLAPS) has been designed as an interactive proof assistant for deductively verifying TLA$^+$ specifications. Designed to be independent of any particular theorem prover, it is based on a hierarchical proof language. A proof manager interprets this language and generates proof obligations corresponding to leaf proofs, which can be discharged by different back-end provers. The current release, restricted to non-temporal reasoning, includes Isabelle/TLA$^+$, an encoding of the set theory underlying TLA$^+$ as an object logic in the proof assistant Isabelle, the tableau prover Zenon, and a back-end for SMT solvers.

This article will first give an overview of the overall design of TLAPS and its proof language and then focus on proof certification in TLAPS. Since the use of different back-end provers raises legitimate concerns about the soundness of the integration, we expect back-end provers to produce proofs that can be checked by Isabelle/TLA$^+$, our most trusted back-end, and this is currently implemented for the Zenon back-end. I will review our experiences with proof certification, and to what extent it has contributed to avoiding soundness bugs, and will indicate future work that we intend to carry out in order to improve our confidence in the soundness of TLAPS.

1 The TLA$^+$ proof language

TLA$^+$ [4] is a specification language originally designed for specifying concurrent and distributed systems and their properties. It is based on Zermelo-Fraenkel set theory for modeling data structures and on the linear-time temporal logic TLA for specifying system executions and their properties. The TLA$^+$ proof system TLAPS [2, 3] has been designed as an interactive proof assistant for deductively verifying TLA$^+$ specifications. Designed to be independent of any particular theorem prover, it is based on a hierarchical proof language.

As a simple example of the TLA$^+$ proof language, let us consider the proof of Cantor's theorem. Given the definitions[1]

$$\begin{array}{l} \mathit{Range}(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\} \\ \mathit{Surj}(f, S) \triangleq S \subseteq \mathit{Range}(f) \end{array}$$

of the range of a function and the notion of a function f being surjective for set S, Cantor's theorem can be stated as

$$\text{THEOREM } \mathit{Cantor} \triangleq \forall S : \neg\exists f \in [S \to \text{SUBSET } S] : \mathit{Surj}(f, \text{SUBSET } S)$$

[1]The application of function f to argument x is written as $f[x]$ in TLA$^+$. The TLA$^+$ expression $[S \to T]$ denotes the set of functions whose domain is S and such that $f[x] \in T$ for all $x \in S$.

```
THEOREM Cantor ≜ ∀S : ¬∃f ∈ [S → SUBSET S] : Surj(f, SUBSET S)
PROOF
⟨1⟩1. ASSUME NEW S,
                 ∃f ∈ [S → SUBSET S] : Surj(f, SUBSET S)
      PROVE  FALSE
  ⟨2⟩. PICK f ∈ [S → SUBSET S] : Surj(f, SUBSET S)
    OBVIOUS
  ⟨2⟩2. ¬Surj(f, SUBSET S)
    ⟨3⟩1. DEFINE D ≜ {x ∈ S : x ∉ f[x]}
    ⟨3⟩2. D ∈ SUBSET S
      OBVIOUS
    ⟨3⟩3. D ∉ Range(f)
      BY DEF Range
    ⟨3⟩4. QED
      BY ⟨3⟩2, ⟨3⟩3 DEF Surj
  ⟨2⟩3. QED
    BY ⟨2⟩2
⟨1⟩2. QED
  BY ⟨1⟩1
```

Figure 1: A hierarchical proof of Cantor's theorem in TLA$^+$.

where SUBSET S denotes the powerset (the set of all subsets) of S. A hierarchical proof of that theorem appears in Figure 1. Proof steps carry labels of the form $\langle d \rangle n$, where d is the depth of the step (which is also indicated by indentation in the presentation of the proof), and n is a freely chosen name—in the example, steps at any given level are just numbered. There are many different ways to write any given proof; in fact, Cantor's theorem could simply be proved by one of the automatic proof backends available in TLAPS. The user chooses how to decompose a proof, starting a new level of proof that ends with a QED step establishing the assertion of the enclosing level. Assertions appearing at the leaves of the proof tree are considered trivial enough to be passed to the automatic provers of TLAPS.

Step $\langle 1 \rangle 1$ reformulates Cantor's theorem as a sequent (TLA$^+$ sequents are written as ASSUME ... PROVE ...) that assumes given some set S for which there exists a surjective function from S to SUBSET S, in order to prove a contradiction. Step $\langle 1 \rangle 2$ deduces the assertion of the theorem from that sequent. (In general, proofs of QED steps are written before the proofs of the steps that precede it.) The level-2 proof starts by picking some such surjective function f and then shows that it cannot in fact be surjective—hence $\langle 2 \rangle 3$ deduces a contradiction. The main argument of the proof appears in the level-3 step, which defines the diagonal set $D \subseteq S$, which can by definition not be in the range of f.

The proof of Cantor's theorem shows some of the main features of the TLA$^+$ proof language. Hierarchical structure is the key to managing complexity. All proof obligations that appear in a proof are independent of one another and can be elaborated and verified in any order. In general, facts and definitions to be used to establish leaf proof steps must be cited explicitly in order to keep the search space for the automatic proof backends manageable.

In our experience, the proof language scales well to large proofs. TLAPS is integrated into

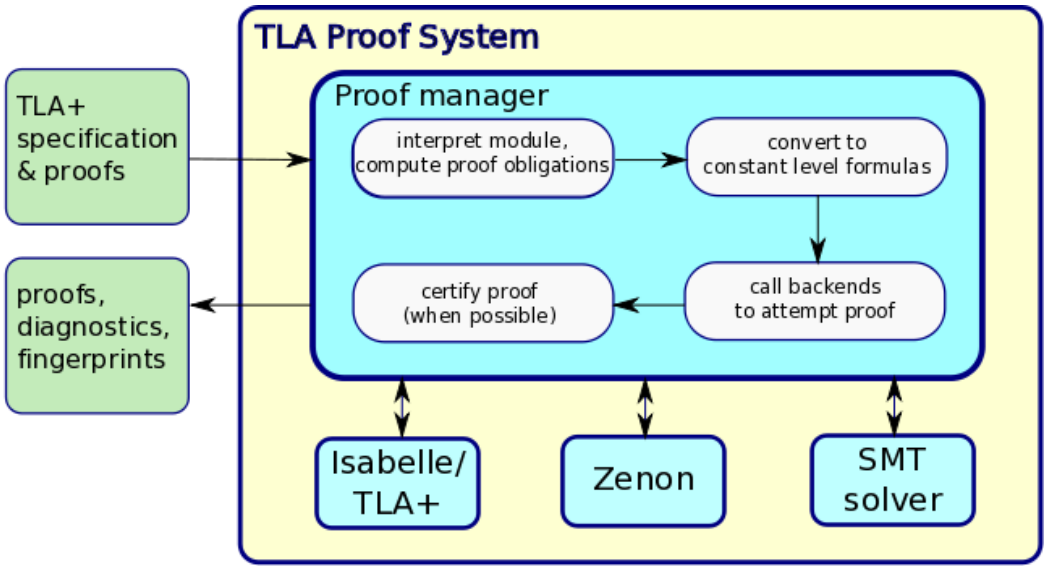

Figure 2: Architecture of TLAPS.

the TLA+ Toolbox, an Eclipse-based IDE that provides features to help reading and writing hierarchical proofs. In particular, proof levels can be hidden or expanded selectively, and the results of proof checking are indicated by color coding. The prover also maintains a database of fingerprints of proof obligations in order to remember what it has already proved previously—even if the proof has been reorganized, or if a step has been deleted and reinserted.

2 Proof checking in TLAPS

Figure 2 shows the basic architecture of the system. The central part of TLAPS is the proof manager, which interprets the proof language and generates proof obligations corresponding to leaf proofs. These can be dispatched to different backend provers. The current release [5], restricted to non-temporal reasoning, includes Isabelle/TLA+, an encoding of the set theory underlying TLA+ as an object logic in the proof assistant Isabelle, the tableau prover Zenon [1], and a backend for SMT solvers.

The proof language is designed to be independent of any particular backend prover. For example, users indicate which definitions to expand and which facts to use, but they cannot tell the backends how to use any given fact—such as for forward or backward chaining, or for rewriting. The set of backends integrated into TLAPS is therefore open-ended: it is enough to implement a translation from (a fragment of) TLA+ to the input language of the backend, and an interface to call the backend with the input corresponding to a proof obligation and retrieve the result. However, the use of different backends, based on different logical theories, raises legitimate concerns about the soundness of their joint use. TLAPS therefore provides a facility for proof certification. Whenever possible, we expect backend provers to produce a proof trace that can be checked by Isabelle/TLA+, our most trusted backend. This is currently implemented for Zenon, which can produce a proof script in the Isar language that is checked by Isabelle.

In our experience, the overhead of proof checking is insignificant—all the more so because

users will check the proof only at the very end, after proof development has finished. The proofs that Zenon produces are detailed and can be checked without any significant proof search. We suspect that checking SMT proofs could become more problematic because SMT solvers can handle much larger formulas that may lead to correspondingly larger proof traces. Techniques for compressing SMT proofs by transforming them in order to avoid redundant steps are then likely to be of interest.

We have yet to find our first Zenon bug. However, certain transformations carried out by the proof manager are critical for soundness, and these should also be checked. The operations of the proof manager fall into two categories. First, it maintains the context of the current proof step, including the symbols that are currently declared and the facts and hypotheses that can be used. The proof language has a block structure with clear scoping rules, and therefore errors in this part can in general be avoided by enforcing a clear program structure.

Secondly, the proof manager is responsible for eliminating references to state variables in (non-temporal) TLA$^+$ formulas. State variables appear in formulas describing state and transition predicates, such as

$$x > c \qquad \text{or} \qquad x > c \land x' = x - 1$$

where x is a state variable and c is a constant.[2] Such formulas are evaluated over pairs of states, with the convention that unprimed variable occurrences denote the value of the variable in the first state, and primed occurrences in the second state. If P is a state predicate (i.e., a formula without free primed state variables), the notation P' refers to a copy of P where all state variables have been replaced by their primed versions, so $(x > c)'$ is $x' > c$. Similarly, for any transition predicate A, the notation ENABLED A denotes A with existential quantification over all primed state variables, so[3]

$$\text{ENABLED } (x > c \land x' = x - 1) \;\equiv\; \exists x' : x > c \land x' = x - 1$$

Since backend provers implement standard first-order logic and do not know about state variables, they handle x and x' as two unrelated first-order variables, and the proof manager must expand notation such as the above. Although this is not difficult as long as definitions are completely expanded, it is easy to get the translation wrong in the presence of unexpanded definitions, in the case of more complicated expressions such as (ENABLED $A)'$, or when expressions appear in argument positions of operators. The soundness of these transformations should therefore be checked by the trusted backend Isabelle/TLA$^+$ of TLAPS, and indeed we have already seen errors in our implementations of these operations. However, this requires a proper definition in Isabelle not only of the set theory underlying TLA$^+$, but also of the notions of state, state variables, and state and transition expressions.

3 Conclusion

TLAPS is based on a hierarchical proof language that results in a clear proof structure and enables independent checking of the different proof obligations that arise in a proof. From a logical point of view, TLA$^+$ proof trees correspond to natural-deduction proofs. The language is interpreted by the proof manager that tracks the context, computes the proof obligations, and translates them to the input formats of backend provers. TLAPS provides an option for retrieving proof traces produced by the backends and having them certified by Isabelle/TLA$^+$,

[2] The category of each symbol appearing in a TLA$^+$ module is declared in the header part of the module.

[3] Semantically, the state predicate ENABLED A holds in a state s if there is some state t such that A holds over the pair (s, t).

a faithful encoding of the TLA$^+$ set theory as an object logic in the proof assistant Isabelle. This is currently implemented for the tableau prover Zenon, and we plan to extend proof certification to the SMT backend. However, in our experience, it is much more likely that a proof is meaningless because of some error in the underlying specification than because of an error in one of the backends. Moreover, the proof manager implements some transformations that are critical for the overall soundness of the system, and these are currently not checked. In the near future, we plan to extend TLAPS for supporting temporal reasoning, and in particular the verification of liveness properties, and this will pose new challenges for proof certification.

Acknowledgement The development of TLAPS is a joint effort carried out at the Joint Microsoft Research-INRIA Centre in Saclay. Kaustuv Chaudhuri, Denis Cousineau, Damien Doligez, Leslie Lamport, and Hernán Vanzetto contributed to different aspects of the work reported here.

References

[1] R. Bonichon, D. Delahaye, and D. Doligez. Zenon : An extensible automated theorem prover producing checkable proofs. In N. Dershowitz and A. Voronkov, editors, *LPAR*, volume 4790 of *LNCS*, pages 151–165, Yerevan, Armenia, 2007. Springer.

[2] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz. Verifying safety properties with the TLA$^+$ proof system. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 142–148, Edinburgh, UK, 2010. Springer.

[3] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto. TLA$^+$ proofs. In D. Giannakopoulou and D. Méry, editors, *Formal Methods*, LNCS, Paris, France, 2012. Springer. To appear.

[4] L. Lamport. *Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2003.

[5] The TLA Proof System. Web page. `http://msr-inria.inria.fr/~doligez/tlaps/`.