Higher-order Automation in TLAPS

(Work in progress)

Antoine Defourné, INRIA
Petar Vukmirovic, VU Amsterdam

TLA$^+$, as a logical framework, is often presented as set theory in *first-order* logic [Lam02]. This is not strictly true, as some features of TLA$^+$ can reasonably be called "second-order". The dedicated prover of TLA$^+$, TLAPS [CDL$^+$12], relies on backend solvers, most of which only support first-order logic. Isabelle is the only exception. Thus there is a class of problems that are currently very difficult for TLAPS to handle. This works aims at bridging this gap, by extending TLAPS with a higher-order backend solver, namely Zipperposition.

Zipperposition is a superposition-based automatic theorem prover envisioned as a vehicle for prototyping various extensions to superposition calculi. Its support for higher-order is based on a complete calculus for extensional polymorphic clausal higher-order logic [BBT$^+$19]. Recent pragmatic extension to full higher-order logic [Vuk20] and an improved higher-order unification procedure [VBN20] were the main factors that contributed to Zipperposition winning the higher-order division of CASC-J10 theorem proving competition [Sut20].

TLA$^+$ has several second-order features. To illustrate a few, we turn to a simple formalization. The goal is to define the sum over a series:

$$\sum_{i=1}^{n} s_i$$

That expression has two parameters: the natural number n, and the term s. The variable i is bound in s_i; thus, there is a lambda-abstraction "$\lambda i.\, s_i$" hidden behind the notation. TLA$^+$ allows the declaration of second-order operators, that is, operators that take first-order operators as arguments. We can then represent the parameter s by an operator $S(\_)$ given as argument to sum.

We want to define the sum by recursion on n. Module NaturalsInduction provides utilities to define recursive *functions* on Nat, but not recursive operators. This implies that we cannot represent n as an argument of the sum operator. Instead, we must define $\mathsf{sum}(S)$ as a TLA$^+$ function on Nat, so that the full expression will be $\mathsf{sum}(S)[n]$ (assuming $n \in \mathsf{Nat}$).

```
EXTENDS TLAPS, Naturals, NaturalsInduction

(** For all operators S(_) and n ∈ Nat, sum(S)[n] is
    the sum of all S(i) for 0 <= i < n *)
sum(S(_)) ==
    LET sumRec[m ∈ Nat] ==
        IF m = 0 THEN 0 ELSE S(m) + sumRec[m - 1]
    IN
    sumRec

THEOREM SumDefConclusion ==
    ASSUME NEW S(_)
    PROVE NatInductiveDefConclusion(sum(S), 0, LAMBDA v,n : S(n) + v)
    OMITTED
```

```
16 THEOREM SumDef ==
17     ASSUME NEW S(_), NEW n ∈ Nat
18     PROVE sum(S)[n] = IF n = 0 THEN 0 ELSE S(n) + sum(S)[n - 1]
19     (* Isabelle fails. SumDefConclusion is a lemma that needs to be
20        instanciated with the operator S(_) *)
21     BY SumDefConclusion DEF NatInductiveDefConclusion
```

The definition of sum in lines 5-9 is not enough by itself, because TLA$^+$ does not guarantee the recursive function exists. The existence of a function that matches the definition must be proven manually. Module NaturalsInduction provides generic theorems for this. By following a simple pattern of theorems (given at the end of the module's source code), we can recover the basic facts we need about sum.

Theorem SumDefConclusion essentially states that a function matching the recursive definition exists. This is expressed by the predicate NatInductiveDefConclusion—its precise definition does not matter to us. Next, theorem SumDef also states that sum matches the intended definition, but in a form that is more practical to us. Unfortunately, TLAPS fails to prove that theorem.

To prove the goal, one has to first instantiate SumDefConclusion with the operator $S(\_)$, and then finish the proof with the definition of NatInductiveDefConclusion. This instantiation step is problematic, because the instance is a first-order operator. It is thus *second-order* reasoning. Currently Isabelle is the only backend that is able to perform this kind of reasoning, which makes the proof script very fragile.

So far, our work has lead us to extend TLAPS with an export to the TPTP language, a standard input format for automatic provers, and use this export to make Zipperposition solve TLA$^+$ proof obligations. Despite this being a work-in-progress, we managed to prove theorem SumDef with Zipperposition.

References

[BBT+19] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Superposition with lambdas. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2019.

[CDL+12] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA+ Proofs. In Dimitra Giannakopoulou and Dominique Méry, editors, *18th International Symposium On Formal Methods - FM 2012*, volume 7436 of *Lecture Notes in Computer Science*, pages 147–154, Paris, France, August 2012. Springer. The original publication is available at www.springerlink.com.

[Lam02] Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[Sut20] Geoff Sutcliffe, editor. *Proceedings of the 10th IJCAR ATP System Competition (CASC-J10)*, July 2020.

[VBN20] Petar Vukmirovic, Alexander Bentkamp, and Visa Nummelin. Efficient full higher-order unification. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Vuk20] Petar Vukmirovic. Boolean reasoning in a higher-order superposition prover. In *7th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, 2020.