Higher-order Automation in TLAPS
Antoine Defourné, Petar Vukmirovic

To cite this version:
Antoine Defourné, Petar Vukmirovic. Higher-order Automation in TLAPS. TLA+ Community Event 2020, Oct 2020, Virtual, France. hal-02990614

HAL Id: hal-02990614
https://hal.science/hal-02990614
Submitted on 15 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 work 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 \( \text{sum} \).

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

```
** EXTENDS TLAPS, Naturals, NaturalsInduction **

(** For all operators \( S(\_) \) and \( n \in \texttt{Nat} \), \( \text{sum}(S)[n] \) is 
the sum of all \( S(i) \) for \( 0 \leq i < n \) *)

\[ \text{sum}(S(\_)) == \]

LET \[ \text{sumRec}[m \in \texttt{Nat}] == \]

IF \( m = 0 \) THEN 0 ELSE \( S(m) + \text{sumRec}[m - 1] \)

IN \( \text{sumRec} \)

THEOREM SumDefConclusion ==

ASSUME NEW \( S(\_) \)

PROVE \( \text{NatInductiveDefConclusion}(\text{sum}(S), 0, \text{LAMBDA } v, n : S(n) + v) \)

OMITTED
```
THEOREM SumDef ==

ASSUME NEW S(_), NEW n ∈ Nat
PROVE sum(S)[n] = IF n = 0 THEN 0 ELSE S(n) + sum(S)[n - 1]

(* Isabelle fails. SumDefConclusion is a lemma that needs to be
instantiated with the operator S( _) *)

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


