Challenge Examples for Higher-Order ATPS

Chad E. Brown

March 29, 2019

We give three challenge examples for higher-order automated theorem provers like Satallax [2]. In each case we discuss how Satallax could communicate with an advisor to direct it in a way to find the proof and speculate about how such an advisor might be trained using machine learning techniques. The advisor used here was hard coded to work for these specific examples.

1 Replacement Implies Separation

The first example is the proof that the separation axiom in set theory follows from the replacement axiom. This is the first scheme proven in [3] and is not difficult for a human. Unfortunately it requires a higher-order instantiation currently out of reach for higher-order automated theorem provers. There is currently no mode with which Satallax has been able to prove this example.

Let $\in: \iota \to \iota \to o$ be a constant which we will write in infix. The replacement property can be stated is follows:

$$\forall A_\iota.\forall r_{\iota\to\iota\to o}.(\forall x.x \in A \to \forall y z_\iota.r\,x\,y \wedge r\,x\,z \to y = z) \to \exists B_\iota.\forall y.y \in B \Leftrightarrow \exists x.x \in A \wedge r\,x\,y.$$

The separation property can be stated as follows:

$$\forall A_\iota.\forall p_{\iota\to o}.\exists B_\iota.\forall x.x \in B \Leftrightarrow x \in A \wedge p\,x.$$

While Satallax cannot currently prove separation from replacement on its own, it can easily prove it by receiving advice from the hard-coded advisor:

```
time satallax -m mode300 -advisorsocket 2332 replimpsep.p
Connected to 2332
% SZS status Theorem
% Mode: mode300
% Inferences: 242

real 0m0.127s
user 0m0.008s
sys 0m0.004s
```

The advisor handles this example as follows. When the replacement axiom is processed, it is recognized and the type ι and the constant $\in$ are remembered. As search proceeds the negated conjecture will lead to a proposition of the form

$$\forall B.\neg(\forall x.x \in B \Leftrightarrow x \in A \wedge p\,x)$$

for eigenvariables A and p. The advisor recognizes this formula and remembers the A and p. After both of the propositions have been recognized, the advisor pushes the general suggestion of using the instantiation $\lambda xy.px \wedge x = y$ onto its suggestion stack. This suggestion is given to Satallax the next time Satallax requests a general suggestion.

While giving this suggestion is clearly the most helpful advice, the rest of the proof is still not completely trivial. After the suggestion has been given the advisor recognizes future propositions which are known to be part of a proof and gives these a high priority (and all other propositions a low priority). In addition, once the existential quantifier in the replacement property has generated an eigenvariable as a witness, this eigenvariable is explicitly suggested by the advisor as an instantiation, as this will be the witness for the separation property. This, of course, makes the proof easy for Satallax.

In this case, it is not clear how such an advisor should be learned from examples. Presumably there would need to be other examples which make successful use of an instantiation of the form $\lambda xy.px \wedge x = y$.

2 Injective Cantor

The injective form of Cantor's Theorem was given as a challenge problem in [1] along with a suggested idea for a proof [1]. It can be stated as follows:

$$\neg\exists f_{(\iota\to o)\to\iota}.\forall XY_{\iota\to o}.f\,X = f\,Y \to X = Y.$$

Unlike the surjective form of Cantor's Theorem, the injective version seems to require a nontrivial instantiation and clever choices after this instantiation has been made. As discussed in [1] considering a diagonal set of the form $\{f\,Y|\neg Y\,(f\,Y)\}$ leads to a contradiction. However, representing this set in simple type theory requires the use of a higher-order quantifier *inside* the higher-order instantiation. For example, the diagonal set can be represented as follows:

$$D := \lambda x_\iota.\exists Y_{\iota\to o}.x = f\,Y \wedge \neg Y\,x.$$

Generating such an instantiation by blind enumeration seems unlikely and it is not clear how a learning algorithm would be encouraged to suggest it. Even once we have the instantation, a cut-free proof would require some unintuitive steps.[1] The more intuitive

[1] I encourage the reader to try this. The only assumption you have is injectivity of f. You are allowed to use D but no cuts. What do you do? I know a way to proceed, shown to me by Peter Andrews, but you have to do something that seems like it is obviously a bad idea. Ask me if you want to know what I mean.

step would be to simply give $D\,(f\,D)$ as a cut formula (as is more or less suggested in [1]).

The hard-coded advisor recognizes when a proposition asserting a name of a type like $(\alpha \to o) \to \alpha$ to be injective is processed. If such a proposition is recognized for a name f, the term D above is constructed. Instead of simply suggesting this as an instantiation, the advisor first suggests $D\,(f\,D)$ as a cut formula and then suggests D and $f\,D$ as instantiations. Even with these suggestions, finding the proof still requires the advisor to block unhelpful paths and to suggest an eigenvariable (coming from the quantifier inside the D) as a useful instantiation. Once enough information was hard-coded into the advisor, the problem became easy.[2]

```
time satallax -m mode300 -advisorsocket 2332 injcantor.p
Connected to 2332
% SZS status Theorem
% Mode: mode300
% Inferences: 13

real 0m0.031s
user 0m0.004s
sys 0m0.000s
```

Again, it is unclear how an algorithm could learn to synthesize either the instantiation

$$D := \lambda x_\iota.\exists Y_{\iota\to o}.x = f\,Y \wedge Y\,x$$

or the cut formula $D\,(f\,D)$. I know of no other example that requires this instantiation.

It is conceivable a learner could start to recognize formulas that appear to say a function f of type $(\alpha \to o) \to \alpha$ is injective and in such cases suggest the D above and the cut formula $D\,(f\,D)$. This could be seen as a human writing a "tactic" and the learner recognizing when to use it. On the other hand, it seems like a more successful approach in such a case would be to include the instance of Injective Cantor for f if f is recognized to be "probably" injective instead of trying to reprove Injective Cantor.

3 Commutativity of Addition

As a final challenge example, we consider commutativity of addition on the natural numbers. This requires a proof by induction that also requires two subinductions. As

[2]The typical process of hard-coding the advisor was to run Satallax with the advisor for a few seconds with both Satallax and the advisor giving verbose output. Using the output, I could check by hand the latest propositions that should be "good" but were labeled by the advisor as "bad" (the default). These propositions were added to the function adapting priorities so they would be recognized as "good." In many cases there were instantiations that also needed to either be suggested or at least recognized as "good." In every case, as soon as Satallax succeeded, I stopped hard-coding, but by that point the problem was typically solved quickly.

a higher-order theorem, this means there will be a higher-order quantifier that must be instantiated in three different ways.

Let $0 : \iota$, $s : \iota \to \iota$ and $a : \iota \to \iota \to \iota$ be constants for 0, successor and addition. We will write $u + v$ for $a\,u\,v$. Assume the induction principle:

$$\forall P_{\iota \to o}.P\,0 \to ((\forall x.P\,x \to P\,(s\,x)) \to \forall x.P\,x)$$

Furthermore assume two axioms defining a:

$$\forall y.0 + y = y$$

and

$$\forall xy.(s\,x) + y = s\,(x + y).$$

The conjecture we wish to prove is $\forall x.\forall y.x + y = y + x$.

If we were proving this in an interactive theorem prover, a reasonable approach is to prove two lemmas by induction:

$$\forall x.x + 0 = x$$

and

$$\forall xy.x + (s\,y) = s\,(x + y)$$

and then use these two lemmas to prove commutativity. An advisor might suggest these lemmas as cut formulas. The current hard-coded advisor does not do this, but instead inlines the subinductions when required.

The hard-coded advisor recognizes the induction axiom for a type ι, a constant 0 and a unary function s and remembers it. If it then sees a proposition of the form $\neg\forall y.c + y = y + c$ being processed, it remembers the addition symbol and the name c (an eigenvariable in this particular proof). After seeing both the induction axiom and the negation of the half quantified commutativity formula, the hard-coded advisor begins to make the following suggested instantiations of type $\iota \to o$: $\lambda y.c + y = y + c$ and $\lambda x.x + 0 = x$. It also suggests instantiations 0 and c of type ι. After this the advisor begins to adapt the priorities of propositions, instantiations and confrontations (equational steps) to keep the search as directed as possible.

After instantiating the induction property with $\lambda y.c + y = y + c$, a subformula $\neg\forall x.c + x = x + c \to c + (s\,x) = (s\,x) + c$ will eventually be processed. As a consequence an eigenvariable, which we call d, will be generated. After this eigenvariable has been generated a new higher-order instantiation $\lambda x.x + (s\,d) = s\,(x + d)$ (corresponding to the other subinduction) will be suggested, along with instantiations d and $s\,d$ of base type. Along the way certain other eigenvariables are generated and must be suggested as instantiations.

For the most part the advisor proceeds by giving high priority to formulas it explicitly recognizes. However, if the formula is an equation or disequation where each side has at most one addition operator, at most two occurrences of s and at most two

occurrences of 0, then it is also given high priority. Instantiations are given high priority if they either names (including eigenvariables and 0) or the successor of a name.[3]

Once this is done, `mode1` with help from the advisor can prove the theorem in about a second.

```
time satallax -m mode1 -advisorsocket 2332 addcom.p
Connected to 2332
% SZS status Theorem
% Mode: mode1
% Inferences: 2600

real 0m1.208s
user 0m0.108s
sys 0m0.116s
```

Proofs by induction are typically hard for higher-order automated theorem provers, but this case in which three inductions must be done is far out of reach of current procedures. It's conceivable that one could have a collection of induction proofs easy enough for Satallax to do, but it is unclear how it could learn from those proofs to build an advisor capable of directing Satallax to prove commutativity of addition.

4 Conclusion

The hard-coded advisor demonstrates that it is possible to take information Satallax generates during search and direct it in a way to obtain proofs that are otherwise out of reach. The real challenge is to use machine learning to automatically generate the advisor using data from successful searches.

In the three examples, it is difficult to see how this could be done. Fortunately, this is more of a challenge for machine learning than automated theorem proving, so it is not necessary for me to see how it could be done. I can simply pose it as a challenge.

References

[1] Peter B. Andrews, Matthew Bishop, and Chad E. Brown. System Description: TPS: A Theorem Proving System for Type Theory. In *Automated Deduction - CADE-17*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 164–169. Springer-Verlag, 2000.

[2] Chad E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. *Journal of Automated Reasoning*, pages 57–77, 2013.

[3] Keep in mind there is a difference between the advisor *suggesting* an instantiation and the advisor adapting priorities of an instantiation Satallax has generated.

[3] Library Committee. Boolean properties of sets — definitions, April 2002.