Remarks about Satallax and Machine Learning

Chad E. Brown

August 8, 2018

At a high level here is how machine learning could, in principle, help an automated prover like Satallax [3]: When Satallax is searching for a proof, it could optionally pass messages to a "learner." The learner could process this information and use it to help construct an "advisor." In later runs, Satallax could optionally pass messages to the advisor and the advisor could give advice on how the search should proceed.

We briefly break the fourth wall. In Summer 2018 I added some code to a branch of Satallax that connects to a learner and/or an advisor via a socket and passes relevant messages during the search. This framework will allow people to experiment with various machine learning tools that need not be coded in ocaml. I also hard coded an advisor that can direct Satallax to find proofs of some very challenging examples where there is very little hope of Satallax finding the proofs on its own. The hard coded advisor makes it clear that there is a sufficient amount of information being communicated to be helpful. It is unclear how such an advisor could be learned, and I will leave this to others to investigate. The learning code Färber implemented directly in Satallax [6] could be extracted to be a "learner" and "advisor" using this framework, but there is no real benefit to doing this, since he wrote it in ocaml anyway. Now I will go back to the normal pattern of writing "we." For now.

In Section 1 we document the messages being passed from Satallax to a learner with comments about what a learner might be expected to do with these messages. In Section 2 we document the messages being passed between Satallax and the advisor. In Section 3 we discuss five examples Satallax can prove with the advice of a hard-coded advisor. Three of these examples require non-trivial higher-order instantiations that are suggested directly by the advisor, and one example also makes use of a cut formula suggested by the advisor. A significant challenge would be to find a collection of problems from which Satallax and a learner could extract an advisor capable of solving these same examples.

1 Communicating with a Learner

If Satallax is given arguments "`-socketlearner` n" then whenever it begins to try to prove a subgoal, it attempts to connect to a local socket at port n and send messages during the search. The connection is disconnected after each subgoal is either proven

or Satallax has given up. A single problem usually has one "subgoal" (to prove the conjecture from the axioms), but under some flag settings Satallax will split one problem into multiple subgoals.

Communication with a learner flows in only one direction: from Satallax to the learner. Each message is sent by sending a byte identifying the type of message (indicated below for each message) possibly followed by some serialized data. For the details of the serialization of integers, strings, types and terms the reader can inspect the code. Note that atoms and literals refer to integers that Satallax associates with closed propositions generated during the search. As is usual in higher-order, propositions are special cases of terms.

The following messages are sent when Satallax inserts a search option into the priority queue. Not every search option leads to sending a message. In particular, enumeration-related options are not communicated to the learner.

4 When Satallax inserts a `ProcessProp1` search option with priority p and proposition φ, it sends the learner this message along with the integer p and term φ.[1]

5 When Satallax inserts a `ProcessProp2` search option with priority p and proposition φ, it sends the learner this message along with the integer p and term φ. This might never happen. `ProcessProp2` was somewhat experimental and I think it still is.

6 When Satallax inserts a `Mating` search option with priority p, literals l_1 and l_2 and corresponding propositions $ps_1 \cdots s_n$ and $\neg pt_1 \cdots t_n$, it sends the the learner this message along with integers p, l_1, l_2 and n followed by the terms $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$.

7 When Satallax inserts a `Confront` search option with priority p, literals l_1 and l_2 with corresponding propositions $s_1 =_\alpha s_2$ and $t_1 \neq t_2$, it sends the learner this message along with integers p, l_1 and l_2, the simple type α and the terms s_1, s_2, t_1 and t_2.

8 When Satallax inserts a `NewInst` search option with priority p and a term s of type α, it sends the learner this message along with p, α and s.

The next messages provide helpful auxiliary information to the learner.

11 If x is the name of an eigenvariable generated to correspond to the formula $\forall_\alpha.s$ (meaning that if $\forall_\alpha s$ is false, then sx will be false), then when x is created Satallax sends this message along with the integer l_1 (where l_1 is the atom for $\forall_\alpha s$), the integer l_2 (where l_2 is the literal for $\neg(sx)$ after normalization), the type α and the string x. This allows the learner to distinguish between different eigenvariables (Skolem constants).

[1] I think this was the only kind of search option used for learning described in [6].

12 Everytime Satallax creates an atom a for a proposition φ, it sends the learner this message with a and φ.

13 When Satallax adds a unit clause as an assumption for minisat (these correspond to the axioms and negated conjecture in the higher-order problem), this message is sent along with the single literal l in the unit clause.

14 When Satallax adds a clause $l_1|\ldots|l_n$ to the set of clauses for minisat, it sends the learner this message with the $n+1$ integeras n, l_1, …, l_n.

15 When Satallax instantiates a formula $\forall_\alpha s$ with a term t of type α, it sends this message along with the litaral for $\forall_\alpha s$, the literal for st (after normalization), the type α and the term t.

Finally two messages communicate the succesful end to a search:

16 The propositional clauses have been reported to be unsatisfiable by MiniSat [5].

17 E reported first-order unsatisfiability.

If the learner gets the message that MiniSat reported unsatisfiability, the learner can find an unsatisfiable core of the propositional clauses (e.g., using PicoMus [2]) and use this to determine which formulas, which eigenvariables and which instantiations were actually used in the proof.

2 Communicating with an Advisor

There is a great deal of overlap in the messages from the previous section. However, in this case Satallax often expects a response from the advisor and will wait to receive the expected response. Other than this, we have the following differences: there are messages to request and send general suggestions and there are no messages to communicate success to the advisor.

During each step when Satallax would ordinarily take the next search option from the priority queue and process it, it first sends the byte **0** to the advisor:

0 The semantics of this message is to ask the advisor if it wants to suggest a new instantiation or a new cut formula. The new instantiation or cut formula is synthesized by the advisor.

This message can be responded to by one of 3 messages from the advisor.

0 The advisor makes no suggestion.

1 This message includes a term φ which should be a well-typed proposition (Satallax will doublecheck this) intended as a "cut" formula or "internal lemma." Satallax creates creates two new `ProcessProp1` search options for φ and $\neg\varphi$ with default priority 0.

2 This message includes a term s which should be well-typed and of some type α (Satallax will check this and construct the α) and process s as a new instantiation of type α (if it is really new).

This is the most significant part of the communication, and for the most interesting examples in Section 3 there would be little hope of finding proofs if there were not these kinds of open ended suggestions.

The following messages are sent when Satallax inserts a search option into the priority queue. In each case Satallax expects a return message with the same first byte (**4**-**8**) followed by an integer (the "adapted priority"). Note that lower values for the priority means the option will be processed more quickly. Negative values are allowed, but values less than -100 are treated as -100.[2]

4 When Satallax inserts a `ProcessProp1` search option with priority p and proposition φ, it sends the advisor this message along with the integer p and term φ. The advisor returns **4** and an adapted priority p'.[3]

5 When Satallax inserts a `ProcessProp2` search option with priority p and proposition φ, it sends the advisor this message along with the integer p and term φ. The advisor returns **5** and an adapted priority p'. This might never happen. `ProcessProp2` was somewhat experimental and I think it still is.

6 When Satallax inserts a `Mating` search option with priority p, literals l_1 and l_2 and corresponding propositions $ps_1 \cdots s_n$ and $\neg pt_1 \cdots t_n$, it sends the the advisor this message along with integers p, l_1, l_2 and n followed by the terms $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$. The advisor returns **6** and an adapted priority p'.

7 When Satallax inserts a `Confront` search option with priority p, literals l_1 and l_2 with corresponding propositions $s_1 =_\alpha s_2$ and $t_1 \neq t_2$, it sends the advisor this message along with integers p, l_1 and l_2, the simple type α and the terms s_1, s_2, t_1 and t_2. The advisor returns **7** and an adapted priority p'.

8 When Satallax inserts a `NewInst` search option with priority p and a term s of type α, it sends the advisor this message along with p, α and s. The advisor returns **8** and an adapted priority p'.

Note that when the advisor suggests a cut or instantiation, Satallax still creates corresponding search options which means new messages (**4** or **8**) will be sent to the advisor to ask about the adapated priorities of these suggestions.

The next messages provide helpful auxiliary information to the advisor. Satallax does not expect a response to these messages and does not wait for a response.

[2] In the hard-coded advisor, I mostly return -1 for "it's good, so do it now" and add 500 to the priority to say, "it's bad, don't do it unless nothing is better."

[3] From inspecting the code, it seems only these priorities are potentially adapted based on the learning described in [6].

11 If x is the name of an eigenvariable generated to correspond to the formula $\forall_\alpha.s$ (meaning that if $\forall_\alpha s$ is false, then sx will be false), then when x is created Satallax sends this message along with the integer l_1 (where l_1 is the atom for $\forall_\alpha s$), the integer l_2 (where l_2 is the literal for $\neg(sx)$ after normalization), the type α and the string x. This allows the advisor to distinguish between different eigenvariables (Skolem constants).

12 Everytime Satallax creates an atom a for a proposition φ, it sends the advisor this message with a and φ.

13 When Satallax adds a unit clause as an assumption for minisat (these correspond to the axioms and negated conjecture in the higher-order problem), this message is sent along with the single literal l in the unit clause.

14 When Satallax adds a clause $l_1|\ldots|l_n$ to the set of clauses for minisat, it sends the advisor this message with the $n+1$ integeras n, l_1, …, l_n.

15 When Satallax instantiates a formula $\forall_\alpha s$ with a term t of type α, it sends this message along with the litaral for $\forall_\alpha s$, the literal for st (after normalization), the type α and the term t.

3 Examples with a Hard-Coded Advisor

We discuss five examples Satallax can prove with the help of a hard-coded advisor. The first one is a very easy higher-order example that Satallax can also prove without help. The second example is first-order and is intended to demonstrate why it is helpful for the learner and advisor to distinguish between different eigenvariables. The last three are very challenging, but simple to describe, higher-order problems. Satallax would have a very difficult time proving any of these three without advice, and it is difficult to see how one would automatically obtain an advisor capable of helping Satallax prove them.

3.1 An Easy Higher-Order Example

Consider the following conjecture:

$$\forall xy_\iota.x \neq y \to \exists p_{\iota\to o}.px \wedge \neg py.$$

Satallax would prove this conjecture by processing the negation of the conjecture and subsequently generated subformulas obtaining eigenvariables a and b for the outer quantifier and breaking down the implication to consider two formulas: $a \neq b$ and (after normalizing logical constants in terms of $=$, $\forall \to$ and $\bot$) $\forall p.pa \to pb$. To continue, an instantiation for the p must be given. In this case the instantiation $\lambda x.x = a$ leads to a solution, and this instantiation is simple enough that it can be found by blind enumeration.

Using the default strategy schedule the first mode Satallax finds a proof with is `mode371`. Examining `mode371` it is clear that its flag settings prefer generating instantiations that involve equality and no other logical constants, and so with this mode, Satallax can find a proof within a second.

Suppose we call satallax with `mode371` while communicating with a learner.

```
time satallax -m mode371 -learnersocket 2323 neqexample.p

Connected to 2323
% SZS status Theorem
% Mode: mode371
% Inferences: 5

real 0m0.023s
user 0m0.012s
sys 0m0.008s
```

Satallax sends about fifty messages to the learner communicating which formulas correspond to which atoms (integers), which eigenvariables correspond to which universal formulas, which instantiations have been used, and so on. The higher-order instantiations generated and communicated to the learner consist of the useful one in two forms: $\lambda x.x = a$ and $\lambda x.a = x$ as well as the useless instantiation $\lambda x.b = a$. The last message indicates that E has found a refutation. A learner could cooperate with E independently of Satallax to find which of the instantiations was used in the proof.

Alternatively we can call Satallax again with the same mode but with the flag `USE_E` set to `false`. In this case the search takes slightly longer and only terminates when MiniSat can determine propositional unsatisfiability. In this case the learner receives about 70 messages. Since the learner has the same set of clauses MiniSat had, the learner can find an unsatisfiabile core, e.g., with PicoMus [2]. In this case there are three propositional clauses corresponding to the three higher-order instantiations $\lambda x.x = a$, $\lambda x.a = x$ and $\lambda x.b = a$. To be specific for those unfamiliar with the interaction of Satallax and MiniSat, there are MiniSat clauses:

```
-5 12
-5 13
-5 14
```

Satallax (and the learner, via messages of type **12**) associates the following atoms with the following formulas:

- 5 corresponds to $\forall p.pa \to pb$.
- 12 corresponds to $a = a \to b = a$.
- 13 corresponds to $b = a \to b = a$.

- 14 corresponds to $a = a \to a = b$.

Furthermore via messages of type **15** the learner knows the instantiations $\lambda x.x = a$, $\lambda x.b = a$ and $\lambda x.a = x$ correspond to the clauses $-5|12$, $-5|13$ and $-5|14$, respectively. PicoMus finds an unsatisfatisfiable core using the clause $-5|12$, but not the clauses $-5|13$ and $-5|14$. The learner can infer from this that $\lambda x.x = a$ is the instantiation that led to a solution.

We can now run Satallax again calling the advisor. The advisor in this case has a hard-coded test watching for a proposition of the form $\forall p.pa \to pb$ with some names a and b. When it sees such a proposition being processed, it adds the instantiation $\lambda x.x = a$ to a stack of suggestions. The next time the advisor is asked for a general suggestion, it suggests this instantiation. Satallax can then easily find the proof, even with modes other than `mode371`. For example using `mode1` without the advisor, Satallax cannot find a proof for the theorem within 30 seconds, but with the the advisor a proof is found quickly.

```
time satallax -m mode1 -advisorsocket 2332 neqexample.p

Connected to 2332
% SZS status Theorem
% Mode: mode1
% Inferences: 7

real 0m0.005s
user 0m0.000s
sys 0m0.000s
```

In this example, it is easy to imagine that a machine learning algorithm could learn to recognize formulas of the form $\forall p.pa \to pb$ and suggest the instantiation $\lambda x.x = a$ in such a case.[4]

3.2 An Example with Many Eigenvariables

We next turn to a first-order example involving many eigenvariables. Let α and β be base types. Let $g : \alpha$, $o : \alpha \to \alpha \to \alpha$ and $r : \alpha \to \beta \to \beta \to o$ be typed names. The idea is that α is a type of binary trees constructed from g and o. For each binary tree, r will give a binary relation on β.

Consider the following propositions:

total-g $\forall x_\beta \exists y_\beta\, r\, g\, x\, y$

[4] Actually, under certain flag settings Satallax will recognize propositions of the form $\forall p.ps \to pt$ with terms s, t not containing p and transform them into $s = t$ to eliminate the higher-order quantifier altogether.

comp $\forall u_\alpha \forall v_\alpha \forall x_\beta \forall y_\beta . r\,(o\,u\,v)\,x\,y \Leftrightarrow \exists z_\beta . r\,u\,x\,z \wedge r\,v\,z\,y$

Intuitively it is easy to see that from these two axioms if we are given any term t of type α constructed exclusively from o and g, then we can prove $\forall x . \exists y . r\,t\,x\,y$. The proofs of these theorems get progressively longer and generate more and more eigenvariables from the **total-g** axiom. All the instantiations of type β will be eigenvariables, and it is important to use the right instantiations. The example is partly intended to demonstrate why it may be important for a learner and advisor to distinguish between eigenvariables from different sources.[5]

Satallax can prove the theorem $\forall x \exists y . r\,(o\,g\,g)\,x\,y$ but already has trouble proving $\forall x \exists y . r\,(o\,(o\,g\,g)\,(o\,g\,g))\,x\,y$. The problem `manyeigens2.p` has the axioms above and the conjecture $\forall x \exists y . r\,(o\,(o\,g\,g)\,(o\,g\,g))\,x\,y$. The hard-coded advisor recognizes a conjecture of the form $\forall x \exists y . r\,(o\,(o\,g\,g)\,(o\,g\,g))\,x\,y$ and then begins watching messages of type `11` to treat different eigenvariables as special. In particular, it recognizes the eigenvariable a_1 from the outermost universal quantifier in the conjecture, and then the next four eigenvariables a_2, a_3, a_4 and a_5 from using **total-g** on the previous eigenvariable. When the advisor is subsequently asked about the priority of instantiations of type β, it gives a high priority of -1 to these first five eigenvariables and a low priority of 500 to all others. When asked about the priority of instantiations of type α, it gives a high priority for g and $o\,g\,g$ and a low priority to all others, simply because I know these are the instantiations required for this particular proof.

Using the advisor and mode `mode371` Satallax can prove the example in about 30s.

```
time satallax -m mode371 -advisorsocket 2332 manyeigens2.p

Connected to 2332
% SZS status Theorem
% Mode: mode371
% Inferences: 83936

real 0m25.678s
user 0m20.456s
sys 0m0.872s
```

Presumably this could be improved, and probably by learning on similar examples. In this case there is a clear collection of theorems to learn from, namely every theorem of the form $\forall x . \exists y . r\,t\,x\,y$ where t is generated from g and o. Since all the proofs follow the same pattern, it is realistic to imagine a learner could construct an advisor that performs well on other theorems of this form.

[5] There are probably better examples. Now that I'm explaining this the example does not seem to generate as many eigenvariables as I originally thought.

3.3 Replacement Implies Separation

We now turn to the first of the three seriously challenging higher-order examples. The first is the proof that the separation axiom in set theory follows from the replacement axiom. This is the first scheme proven in [4] 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 $\in\colon \iota \to \iota \to o$ 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 $\lambda xy.px \wedge x = y$.

3.4 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.[6] The more intuitive 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.[7]

[6] 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.

[7] 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.

```
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.

3.5 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 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.[8]

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

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

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 these last three of the five examples, I cannot see how this could possibly 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.

From a pure automated theorem proving perspective, automatically proving the last three examples looks hopeless. Since the examples are actually not difficult mathematical problems at all, my conclusion is I should stop doing higher-order theorem proving. First-order set theory is calling me again.

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] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT*, 2008.

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

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

[5] Niklas Een and Niklas Sörensson. An extensible sat-solver [ver 1.2], 2003.

[6] Michael Färber and Chad E. Brown. Internal guidance for satallax. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 349–361. Springer, 2016.