Testing Learning from Equational Proofs

Chad E. Brown

Czech Technical University in Prague

Abstract. Keywords:

We describe a set of equational theorem proving problems. In every case the conjecture is provable from the AIM axioms [2]. Indeed we have at least one proof (by a chain of at most 10 equational steps) for each problem. The purpose of this set of problems is to evaluate how well different machine learning techniques can learn to approximate how many equational steps are required to complete a proof. To this end, we define a set of terms and a basic proof calculus using only reflexivity and paramodulation.[1].

1 Terms and Contexts

Let $\mathcal{V}_\forall$ and $\mathcal{V}_\exists$ be two disjoint countably infinite sets of variables. We call the variables $\mathcal{V}_\forall$ *universal variables* and the variables in $\mathcal{V}_\exists$ *existential variables*. We use x, y, z, w, u, v_i to range over universal variables and X, Y, Z, W, U, V_i to range over existential variables. The set $\mathcal{T}$ of terms (where we use s, t to range over terms) is given by the following grammar:

$$x|X|e|(s \cdot t)|(s\backslash t)|(s/t)|T(s,t)|L(s,t_1,t_2)|R(s,t_1,t_2)|K(s,t)|a(t_1,t_2,t_3).$$

We write $\mathcal{V}_\forall(s)$ for the finite set of universal variables that occur in s and $\mathcal{V}_\exists(s)$ for the finite set of existential variables that occur in s

An *equation* is a pair of terms written as $s = t$.

When we are trying to prove an equation, the intended interpretation is $\forall x \cdots \exists X \cdots s = t$ where $x \cdots$ lists the universal variables in s and $X \cdots$ lists the existential variables in t.

We will also assume equations intended to be used to prove new equations. Each of the assumed equations has no universal variables free, but only existential variables. An example is the equation $(X \cdot e) = X$. Since assuming equations is dual to proving equations, the intended interpretation here is $\forall X.(X \cdot e) = X$. While this may seem counterintuitive at first, an easy way to justify the terminology is to not that proving φ follows from $\forall X.(X \cdot e) = X$ is classically equivalent to proving the conjecture

$$(\exists X.(X \cdot e) \neq X) \vee \varphi.$$

[1] The data should be available from http://grid01.ciirc.cvut.cz/~chad/aimleapinfo/

In all of the problems in the set we will assume the same 87 equations. 12 of these equations are loop and AIM axioms [2], 5 of these equations are definiting equations for T, L, R, K and a, and the other 70 are equations that deductively follow from the first 17.

In order to describe paramodulation inferences, we will need the notion of a *context*. Informally, a context is a "term with a hole." We use C to range over contexts. Contexts are generated by the following grammar:

$$\begin{gathered}\bullet|(C \cdot t)|(s \cdot C)|(C\backslash t)|(s\backslash C)|(C/t)|(s/C)|T(C,t)|T(s,C)\\ |L(C,t_1,t_2)|L(s,C,t)|L(s,t,C)|R(C,t_1,t_2)|R(s,C,t)|R(s,t,C)\\ |K(C,t)|K(s,C)|a(C,t_1,t_2)|a(t_1,C,t_2)|a(t_1,t_2,C)\end{gathered}$$

A context C can be combined with a term t to give a term $C[t]$ in an obvious way (by recursively traversing C to put t in the hole $\bullet$).

A *substitution* θ is a partial function sending variables to terms, and $\theta(t)$ is defined in the obvious way.

We say θ is a *unifier of s and t* if $\mathsf{dom}(\theta) \subseteq \mathcal{V}_\exists$ and $\theta(s) = \theta(t)$. We say s and t are *unifiable* if there exists a unifier of s and t.

An *existential renaming* of $s = t$ is $\theta(s) = \theta(t)$ where the domain of θ is existential variables in s and t, $\theta(X) \in \mathcal{V}_\exists$ and $\theta(X) \neq \theta(Y)$ if $X \neq Y$ if $X, Y \in \mathsf{dom}(\theta)$.

2 Equational Provability

Let $s = t$ be an equation and $s_1 = t_1$ be an equation with no universal variables. Assume $(\mathcal{V}_\exists(s) \cup \mathcal{V}_\exists(t)) \cap (\mathcal{V}_\exists(s_1) \cup \mathcal{V}_\exists(t_1)) = \emptyset$. We say $s_2 = t_2$ is a *paramodulant* of $s = t$ using $s_1 = t_1$ if one of the following holds:

- There is a context C and $\mathsf{dom}(\theta) \subseteq \mathcal{V}_\exists$ such that $\theta(s)$ is $C[\theta(s_1)]$, s_2 is $C[\theta(t_1)]$ and t_2 is t. (Informally, we rewrite from left to right on the left.)
- There is a context C and $\mathsf{dom}(\theta) \subseteq \mathcal{V}_\exists$ such that $\theta(t)$ is $C[\theta(s_1)]$, t_2 is $C[\theta(t_1)]$ and s_2 is s. (Informally, we rewrite from left to right on the right.)
- There is a context C and $\mathsf{dom}(\theta) \subseteq \mathcal{V}_\exists$ such that $\theta(s)$ is $C[\theta(t_1)]$, s_2 is $C[\theta(s_1)]$ and t_2 is t. (Informally, we rewrite from right to left on the left.)
- There is a context C and $\mathsf{dom}(\theta) \subseteq \mathcal{V}_\exists$ such that $\theta(t)$ is $C[\theta(t_1)]$, t_2 is $C[\theta(s_1)]$ and s_2 is s. (Informally, we rewrite from right to left on the right.)

If $(\mathcal{V}_\exists(s) \cup \mathcal{V}_\exists(t)) \cap (\mathcal{V}_\exists(s_1) \cup \mathcal{V}_\exists(t_1)) \neq \emptyset$, then we simply say there are no paramodulants of $s = t$ using $s_1 = t_1$ (by convention). The idea is that we only compute paramodulants after ensuring the sets of existential variables are disjoint.

Let $\mathcal{A}$ be a set of equations with no universal variables. Let $s = t$ be an equation. A *paramodulant of $s = t$ (using $\mathcal{A}$)* is any equation $s_2 = t_2$ that is a paramodulant of $s = t$ using $s_1 = t_1$ where $s_1 = t_1$ is an existential renaming of an equation in $\mathcal{A}$. Up to renaming of existential variables, there are finitely many paramodulants if $\mathcal{A}$ is finite.

We define what it means for an equation to be *provable in n steps (from $\mathcal{A}$)* as follows.

- If s and t are unifiable, then $s = t$ is provable in 0 steps.
- $s = t$ is provable in $n+1$ steps if there exists a paramodulant of $s = t$ (using $\mathcal{A}$) provable in n steps.

It is easy to see that provability is stable under renaming variables so we only need to consider one paramodulant up to the choice of existential variables. This makes the search for a proof finitely branching.

3 Proof Search

Let $\mathcal{A}$ be a set of equations with no universal variables. In our dataset this $\mathcal{A}$ is a fixed set of 87 equations. Suppose we wish to check if $s = t$ is provable in n steps from $\mathcal{A}$. An obvious algorithm is the following:

1. If s and t are unifiable, then report success.
2. If $n = 0$, then report failure.
3. Compute a finite set of paramodulants $s_2 = t_2$ from $\mathcal{A}$, obtaining a representative up to renaming of existential variables for each possible paramodulant.
4. Order these paramodulants in some way and for each one ask if $s_2 = t_2$ is provable in $n - 1$ steps from $\mathcal{A}$.

While the algorithm will terminate in principle, it often will not in practice. Estimates based on the dataset show that there are, on average, roughly 500 possible paramodulants at each step. Traversing the entire search space to check if an equation is provable in 10 steps is obviously unrealistic.

If we have an oracle that can accurately estimate how many steps are required to prove $s = t$, then we can use this to order the paramodulants above. Doing this would ensure that if $s = t$ is provable in n steps, then the algorithm will never fail (resulting in backtracking) since the oracle will always choose a paramodulant provable in $n - 1$ (or fewer) steps.

Having a true oracle is unrealistic, but if a function could provide a good estimate of the number of required steps, then the algorithm should require less backtracking. Our goal is to use different techniques to learn such a function and then use the algorithm above to measure how good the estimate given by the function is.

We can force the algorithm to terminate and get information about how much search was required by having a notion of *abstract time*. Informally, the *abstract time* used by a search for a proof is the number of times a paramodulant is chosen in the fourth step of the algorithm.

This algorithm is implemented in the `aimleap` prover. In the implementation the abstract time is simply a counter incremented when a paramodulant is chosen and the recursive call is made and search terminates if a given abstract time limit is reached. The function for computing "distance" from s to t (estimating the number of paramodulation steps required to prove $s = t$) generally works as follows:

- If s is t, then the distance is 0. (Note: this could be generalized to give 0 if s and t are unifiable.)
- Otherwise, if a port to connect to an advisor was given, then ask the advisor to estimate the distance.
- If no port to an advisor was given but some other local function was indicated, then use that function. (For example, we can give a constant to always return as the distance.)

4 The Dataset

We fix $\mathcal{A}$ to be a set of 87 equations as mentioned above. 17 of these are AIM axioms and definitions while the remaining are equations that follow from these AIM equations.

Veroff has obtained a large number of AIM proofs using Prover9 [3]. Analysing some of these proofs it was possible to obtain 3468 equations provable from the equations in $\mathcal{A}$ within 2-10 paramodulation steps. In some cases there were multiple proofs of the same equation, and in most cases it was possible to consider the paramodulation proof in at least two ways (always paramodulating into the left or always paramodulating into the right). As a consequence there were 7066 proofs, with at least one proof for each equation. The length (number of paramodulation steps) in the initial "training" proofs are given in Table 1. (The number of problems adds up to more than 3468 since some equations have proofs of different length and so are counted twice.)

Length	Number of Proofs	Number of Problems
2	3284 (46.5%)	1641 (47.3%)
3	1744 (25.7%)	869 (25%)
4	708 (10%)	353 (10.2%)
5	573 (8.1%)	284 (8.2%)
6	332 (4.7%)	166 (4.8%)
7	168 (2.4%)	83 (2.4%)
8	156 (2.2%)	78 (2.2%)
9	58 (0.8%)	29 (0.8%)
10	32 (0.5%)	16 (0.5%)

Table 1. Lengths of Training Proofs

These pure paramodulation proofs were analyzed to give all the intermediate forms. That is, at each step there was a new goal equation $s = t$. The initial goal never contains existential variables (i.e., one is always proving $\forall x \cdots s = t$), but existential variables may be introduced via paramodulation.

Consider the following example from the data set (p9_9a20c522ba):

$$L(x, e/x, x) = ((x \cdot (e/x))\backslash x)$$

There are two training proofs for this equation, both of which use the following two equations from $\mathcal{A}$:

id2: $((Y/X)\backslash Y) = X$
prop_da958b3f: $L(X\backslash e, X, Y) = ((Y \cdot X)\backslash Y)$

The first training proof paramodulates with id2 from right to left on the left hand side to obtain the intermediate subgoal

$$L((Y/x)\backslash Y, e/x, x) = ((x \cdot (e/x))\backslash x).$$

Note that this subgoal has an existential variable Y and should be interpreted as trying to prove

$$\forall x.\exists Y.L((Y/x)\backslash Y, e/x, x) = ((x \cdot (e/x))\backslash x).$$

The second step of the first proof paramodulates with prop_da958b3f from left to right on the left hand side of the subgoal (sending the Y from the subgoal to e in when unifying) to obtain the new subgoal

$$((x \cdot (e/x))\backslash x) = ((x \cdot (e/x))\backslash x).$$

Since the two sides are the same (hence trivially unifiable), we are done.

The second training proof proceeds in the other direction and does not introduce existential variables. Starting from the initial goal

$$L(x, e/x, x) = ((x \cdot (e/x))\backslash x)$$

we paramodulate with prop_da958b3f from right to left on the right hand side to obtain the subgoal

$$L(x, e/x, x) = L((e/x)\backslash e, e/x, x).$$

We then paramodulate with id2 from left to right on the right hand side to obtain the subgoal

$$L(x, e/x, x) = L(x, e/x, x).$$

Since the two sides are the same, we are done.

From these two training proofs, we know there are one step proofs of

$$L((Y/x)\backslash Y, e/x, x) = ((x \cdot (e/x))\backslash x)$$

and

$$L(x, e/x, x) = L((e/x)\backslash e, e/x, x).$$

We also know there is a two step proof of the initial goal

$$L(x, e/x, x) = ((x \cdot (e/x))\backslash x).$$

By replaying the 7066 proofs of the 3468 equations we obtain data about which top level and intermediate equations are provable in n steps. This data could be used to train a machine learning algorithm to learn to estimate the number of steps required to prove the equation.

5 Initial Results

We have run `aimleap` and other ATPs on the 3468 problems and we summarize the results here. `aimleap` always searched for a proof of length at most 10 (i.e., the depth of the search tree was always bounded at 10) and was always given an abstract time limit of 100.

As a sanity test, an "oracle" advisor was used first. This advisor returns the known distance for goals and subgoals that were seen in the training proofs. For other equations it returned a high distance of 50 (essentially forcing these equations to be pruned from the paramodulant options). We call this the "rote learning" advisor.

As expected, the prover could reprove all 3468 problems given the full rote learned information. Furthermore, no backtracking was required and so the maximum abstract time was 10. In some cases, shorter proofs than the training proofs were found. In 132 cases, new proofs involving only one paramodulation step were found. Table 2 lists the number of equations proven within n steps using this oracle advisor.

Length	Number of Problems
1	132 (3.8%)
2	1912 (55.1%)
3	803 (23.2%)
4	314 (9.1%)
5	147 (4.2%)
6	74 (2.1%)
7	29 (0.8%)
8	29 (0.8%)
9	17 (0.5%)
10	11 (0.3%)

Table 2. Lengths of Proofs using Oracle

To do a cross-validation test, the training data was split into 10 parts. For each part we create a file of distance data to be given to the advisor. For each problem we ran `aimleap` with an advisor using the rote learned distance data from the other 9 parts. We gave an abstract time limit of 100. 800 (21.9%) problems were proven in the cross-validation test.

The next test we tried was to simply give a constant distance. To be more precise, if we ask for the distance between two terms s and t, then we return 0 if s is precisely t and return c otherwise, for a fixed c. The results from trying this with an abstract time limit of 100 and c varying from 0 to 10 is shown in Table 3. A few remarks are in order. In every case except 0, at least the 132 problems with one step proofs were solved. The reason for this is that the one step paramodulant solving the problem is generated as an option at the first step. Since the paramodulant is of the form $s = s$, the distance function will give

the value 0, preferring this option over all other options with distance $c > 0$. After this option is chosen, the proof is complete (in one abstract time step). On the other hand, if $c = 0$, there is no reason to prefer the option $s = s$ over the other options. The only hope would be if the one step option happened to be the first option, and this never happened in practice. Hence using a constant distance function giving 0 solved no problems. A surprising result is that constantly giving a distance of 9 solved just over half the problems. In addition to solving the 132 problems with one step proofs, giving a constant distance of 9 finds many two step proofs. Suppose $s = t$ is a goal with a two step proof. Many paramodulants are generated at the first step and are all estimated to require a 9 step proof. For each paramodulant chosen we get a new subgoal $s_2 = t_2$ and generate its paramodulants. If one of these paramodulants is of the form $s_3 = s_3$, then the estimated distance will be 0, the option will be chosen, and the search succeeds (finding the two step proof). If none of the paramodulants is of this form, then all options will have estimated distance 9. Since we are searching for a proof of at most length 10 and we have already performed one paramodulation, all these options will be filtered and the search will backtrack. In essence, the search proceeds by trying each possible paramodulation and then checking if one more paramodulation will complete the search.

All other constant distance functions performed poorly, solving at most a few more than the problems with one step proofs.

Constant Distance	Number of Problems
0	0
1	135 (3.9%)
2	135 (3.9%)
3	135 (3.9%)
4	135 (3.9%)
5	135 (3.9%)
6	135 (3.9%)
7	135 (3.9%)
8	138 (4.0%)
9	1739 (50.1%)
10	132 (3.8%)

Table 3. Results using Constant Distance

To compare the results with established automated theorem provers, the same 3468 problems were given to Prover9 [3], E [4] and Waldmeister [1]. Each prover was given 60s to solve the problem and default settings were used. E was run with the `-auto-schedule` option. The results are summarized in Table 4. For each of the problems with a known one or two step proof (see Table 2), at least one ATP finds a proof. For 168 the 803 problems (20.9%) with a known three step proof (see Table 2), none of the ATPs finds a proof. It is worth noting that in some cases an ATP did not even find a proof for some of the 132 problems

with a known one step proof. This happened in one case for Waldmeister and in 74 cases for Prover9. E solved all 132 of these problems.

ATP	Problems Solved (60s)
E	2684 (77.4%)
Waldmeister	2170 (62.6%)
Prover9	2037 (58.7%)
At Least One	3079 (88.8%)
All 3	1384 (39.9%)

Table 4. ATP Results

6 To be continued

Next we should try training a variety of machine learning algorithms on the distance data and try `aimleap` with advisors based on the learned functions. We could also try alternative search procedures to `aimleap`. We could also consider extending the dataset and so on.

References

1. Hillenbrand, T., Jaeger, A., Löchner, B.: Waldmeister - Improvements in Performance and Ease of Use. In: Ganzinger, H. (ed.) Proceedings of the 16th International Conference on Automated Deduction. pp. 232–236. No. 1632 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1999)
2. Kinyon, M.K., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: An application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics - Essays in Memory of William W. McCune. Lecture Notes in Computer Science, vol. 7788, pp. 151–164. Springer (2013)
3. McCune, W.: Prover9 and mace4 (2005–2010), `http://www.cs.unm.edu/~mccune/prover9/`
4. Schulz, S.: System Description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Proc. of the 19th LPAR, Stellenbosch. LNCS, vol. 8312. Springer (2013)