Combining Proofs to form Different Proofs

Geoff Sutcliffe
University of Miami, USA

Cynthia Chang
Rensselaer Polytechnic Institute, USA

Deborah McGuinness, Tim Lebo, Li Ding
Rensselaer Polytechnic Institute, USA

Paulo Pinheiro da Silva
University of Texas at El Paso, USA

Abstract

Different Automated Theorem Proving (ATP) systems solve different parts of different problems in different ways. Given a set of proofs produced by ATP systems based on adequately common principles, it is possible to create new proofs by combining proof components extracted from the proofs in the set. It is not generally easy to say that one of the original or new proofs is better or worse than another, but ways to show that two proofs are different are available. This paper describes a process of proof combination to form new proofs that are different from the original set of proofs.

1 Introduction

Proofs form an essential part of mathematics and modern sciences [11]. Proofs allow human users and machines to understand [4, 25] and verify [12, 21] the processes that yield scientific discoveries, logical conclusions, and other forms of results. This work deals specifically with formal proofs generated by Automated Theorem Proving (ATP) systems for first-order logic, but the framework and techniques are applicable to a broad range of proof-like structures, e.g., informal arguments, dataflows, scientific data manipulation, etc. Some examples of such use cases are described in Section 6.

The proofs used in this work come from problems expressed as a set of axioms and a conjecture to be proved, such as those in the Thousands of Problems for Theorem Provers (TPTP)[1] problem library [22]. The proofs are directed acyclic graphs in which the leaf nodes come from the problem, other nodes are inferred from parent nodes, and the root nodes provide an assurance that the conjecture is a theorem of the leaf axioms. Specifically, the ATP systems that have been used produce proofs by contradiction, in which an initial inference step negates the leaf conjecture, all other inferences are at least satisfiability preserving, and the root nodes are $false$. These are typical of proofs found in the Thousands of Solutions from Theorem Provers (TSTP)[2] solution library, which contains solutions to TPTP problems. All the problems and proofs are written in the TPTP language [24].

Given a set of proofs and an equivalence relation between nodes of the proofs, a new proof can be formed by selecting a *replaced* node in a *target* proof, finding an equivalent *replacing* node in a *contributing* proof, and replacing the sub-DAG rooted at the replaced node by the sub-DAG rooted at the replacing node. The resultant *combined* proof is different from the target and contributing proofs. Combining can be iterated in a controlled fashion to generate a series of new combined proofs based on the set of original proofs. The overall process might require that the set of original proofs be proofs for the same problem, or be based on a common background theory – this depends on the application context.

There has been previous work related to this, in the direction of compressing propositional proofs, e.g., [2, 18, 5]. A key difference between that work and this is that here the goal is to

Pascal Fontaine, Aaron Stump (eds.); PxTP 2010, pp. 1-14

[1] `http://www.tptp.org`

[2] `http://www.tptp.org/TSTP/`

produce proofs that are *different* from the original proofs, rather than proofs that are shorter (see Section 3). Further, this work is in the context of first-order logic, which adds extra demands on the process. At a higher level there has been work studying techniques for presenting alternative proofs and proofs at different levels of granularity [3, 1]. The data structure presented in [1] provides a similar functionality to that of the PML data structure described in Section 2. That research was aimed to support proof development in an interactive mathematical proof assistant, and was related to an application in proof explanation and proof adaptation for tutoring systems [19]. Again, the goal here of producing *different* proofs, based on proofs produced by ATP systems, makes this work somewhat distinct.

The rest of this paper is structured as follows: Section 2 describes the PML language, and the translation of TPTP format proofs into PML for the proof combining process. Section 3 describes quantitative measures of proofs, which are used to guide the proof combination process. Section 4 describes the proof combination process, and Section 5 presents an example application of the process. Section 6 concludes.

2 Proofs in PML

The Proof Markup Language (PML) [15] is a semantic web based representation for exchanging explanations, including provenance information that records the sources of knowledge, justification information that describes the steps for deriving the conclusions or executing workflows, and trust information that captures trustworthiness assertions about knowledge and sources. In contrast to the TPTP language, there is less focus on the logical data and the fine-grained reasoning processes - PML supports arbitrary logical data and inference steps including, e.g., extraction of data from non-logical sources, conversion to logical forms, clausification and first-order inferences, etc. The proof combining described in this work is done using PML because of its ability to explicitly represent alternative justifications for a conclusion. The XML-based format also allows for easy parsing and processing.

PML classes are OWL [13] classes, and PML data is therefore expressible in the RDF/XML syntax. PML is used to build OWL documents representing both proofs and proof provenance information. For this work, the representation of proofs is of primary interest. The two main constructs of proofs in PML are NodeSets and InferenceSteps. A NodeSet is used to host a set of alternative justifications for one conclusion. A NodeSet contains:

- A URI that is its unique identifier.
- The conclusion of the proof step.
- The expression language in which the conclusion is written.
- Any number of InferenceSteps, each of which represents an application of an inference rule that justifies the conclusion.

An InferenceStep contains:

- The inference rule that was applied to produce the conclusion.
- The antecedent NodeSets of the inference step.
- Bindings for variables in the inference.
- Any number of discharged assumptions.
- The original sources upon which the conclusion depends.
- The inference engine that performed the inference step.
- A time stamp recording when the inference step was performed.

A proof consists of a collection of NodeSets, with a root NodeSet as the final goal, linked recursively to its antecedent NodeSets.

The translation of a TSTP proof into PML is done by parsing the TSTP file and extracting the necessary information into PML object instances [16]. A proof is translated into a PML NodeSet collection, with each formula in the solution being translated as a singleton member of the collection. Additionally, the conjecture of the corresponding TPTP problem is translated into a PML Query, and the `English` header field of the problem into a PML Question. The Query contains a pointer to the Question and to all NodeSet collections (from different ATP systems) that provide a solution. The Query thus provides a starting point for accessing all the proofs for that problem.

It is important for this work that the PML representation can store multiple justifications (InferenceSteps) for a formula in a proof (a NodeSet). In ATP terms, PML can associate multiple inference steps with a formula, in the sense that the formula is the result of each of the inference steps. This allows a single PML structure to capture multiple proofs, and a single proof can be extracted by choosing one inference step leading to each formula. A limited version of this is used in the proof combining process, described in Section 4. The PML representation is generally convenient for representating alternative derivations, as it is independent of the underlying reasoning process. This independence will be leveraged in future work representing scientific provenance information - see Section 6.

The way that PML captures multiple proofs in one structure is akin to using a reasoning calculus with either a weakening or a juxtaposition rule [7]. In such calculi the "alternative" sub-DAGs leading to a node are combined by the application of one of these rules. In the same way that alternative proofs can be extracted from the PML representation, alternative proofs can be extracted from these richer calculi's proofs, e.g., by using cut elimination on a proof that uses weakening [6].

3 Different Proofs (are Good Proofs)

Section 4 describes how proof combining steps are wrapped in a greedy hill-climbing algorithm. Hill-climbing uses a heuristic function that guides the search to an (locally) optimal solution. For this work the heuristic function evaluates and ranks proofs based on three artifacts:

- the leaf formulae (axioms and conjecture)
- the inferred formulae
- the inference steps (where an inference step is identified with its parent and inferred formulae).

However, as is explained in [23], it turns out to be difficult, if not unreasonable, to directly *rank* proofs according to such artifacts. Instead, it is possible to say only that proofs are *different* from each other. Different proofs may be preferable, depending on the user's point of view. For examples: in mathematics, different proofs can demonstrate different levels of mathematical "elegance", or show that a theorem can be proved from different axiomatic bases; in formal methods, different proofs can be easier or harder to verify, or provide different insights into the structure or process being analyzed; in planning, different proofs can correspond to different sequences of actions that lead to the same goal state; in knowledge based reasoning, different proofs can be built from different knowledge bases, allowing independent agents with different knowledge to reach a common conclusion; in security analysis, different proofs can rely on different ground observations, thus increasing trust in the intelligence information that is

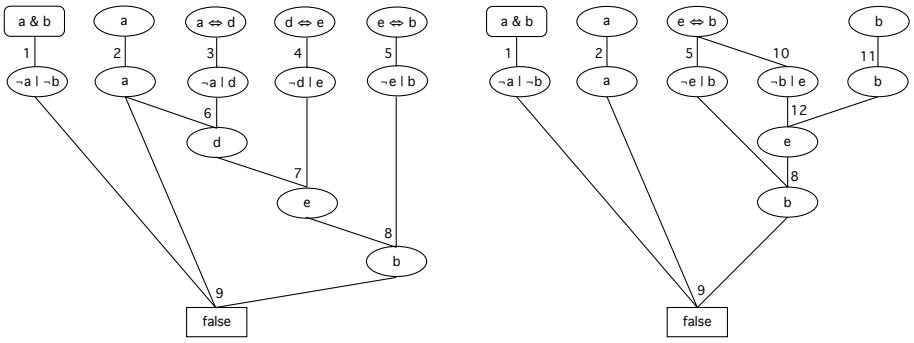

Figure 1: Jaccard Distances

generated. Thus the heuristic function used in this work prefers new combined proofs that are maximally different from other proofs.

Evaluating proofs by *counting* formulae and inference steps is not necessarily meaningful. Comparing the numbers of axioms used in proofs can be meaningless because different axioms contain different amounts of information - using many axioms that each contain a small amount of information is incomparable with using fewer axioms that each contain a large amount of information. Comparing the numbers of inferred formulae or the numbers of inference steps in proofs can be meaningless because of the in-principle differences between inference rules - a proof that uses inference rules that take "large" steps, e.g., hyperresolution, is likely to have less steps and less inferred formulae than one that uses rules that take "smaller" steps, e.g., binary resolution. It is also easy to create proofs with less or more inferences, by combining inferences into larger steps, or splitting inferences into smaller steps. Thus, rather than counting the proof artifacts, it is better to *compare sets* of the artifacts. For this work the Jaccard similarities [10] between sets of proof artifacts are measured. The following propositional problem illustrates this (this example is only for illustration - the problem is trivial):

$Axioms : \{a, b, a \Leftrightarrow d, d \Leftrightarrow e, e \Leftrightarrow b\}$

$Conjecture : a \ \& \ b$

Two different proofs by contradiction are shown in Figure 1. The inferences are numbered for identification. The sets of artifacts – leaf formulae, inferred formulae, and inferences – for the left-hand proof are:

$Leaves : \{a \ \& \ b, a, a \Leftrightarrow d, d \Leftrightarrow e, e \Leftrightarrow b\}$

$Inferred : \{\neg a | \neg b, a, \neg a | d, \neg d | e, \neg e | b, d, e, b, false\}$

$Inferences : \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

and for the right-hand proof:

$Leaves : \{a \ \& \ b, a, e \Leftrightarrow b, b\}$

$Inferred : \{\neg a | \neg b, a, \neg e | b, \neg b | e, b, e, false\}$

$Inferences : \{1, 2, 5, 8, 9, 10, 11, 12\}$

The Jaccard similarity between the sets of leaf formulae is 0.50, between the sets of inferred formulae it's 0.60, and between the sets of inferences it's 0.42.

Given Jaccard similarities between pairs of proofs, a set of proofs is evaluated in terms of the differences between the proofs. The measurement is done using a modified version [23] of the generalized clustering coefficient [14]. This measure views the proofs as vertices of a complete weighted graph, with the Jaccard similarities for a chosen proof artifact as the edge weights. For each pair of edges coincident on a vertex, the pair defines a triplet (of vertices) closed by the edge between the vertices at the other ends of the edges. The weight of a triplet is the average

of the pair of edges' weights. The clustering coefficient C_{sw}[3] is:

$$C_{sw} = \frac{\sum_{all\ triplets} triplet\_weight \times closing\_edge\_weight}{\sum_{all\ triplets} triplet\_weight}$$

For a set of proofs that are all the same, so that all edge weights are 1.0, $C_{sw} = 1.0$. For a set of proofs that are all completely different, so that all edge weights are 0.0, $C_{sw} = 0.0$ (noting that it is necessary to not do the math, to avoid a division by zero). For two proofs, C_{sw} is defined to be the Jaccard similarity between the two sets of chosen artifacts. C_{sw} is undefined for single proofs.

A chosen proof artifact is recorded as a superscript on C_{sw}: Lf for leaf formulae, If for inferred formulae, and Is for inference steps, e.g., C_{sw}^{Lf}. For example, for three (somewhat similar) proofs A, B, and C, with leaf set Jaccard similarities A-$B = 0.66$, B-$C = 0.50$, and A-$C = 0.50$, the triplet formed from the edges A-B and A-C has weight 0.58. Then

$$C_{sw}^{Lf} = \frac{0.29 + 0.29 + 0.33}{0.58 + 0.58 + 0.50} = 0.5482$$

Replacing C with a (quite different) proof D, so that the leaf set Jaccard similarities are A-$B = 0.66$, B-$D = 0.34$, and A-$D = 0.34$,

$$C_{sw}^{Lf} = \frac{0.17 + 0.17 + 0.22}{0.50 + 0.50 + 0.34} = 0.4212$$

For a chosen proof artifact α, a high C_{sw}^{α} indicates that the proofs in the set are highly clustered with respect to α. Conversely a low C_{sw}^{α} indicates that the proofs are rather different from each other with respect to α. Thus for this work D would be preferred over C with respect to leaf sets.

4 Combining Proofs

Reiterating from Section 1, given a set of proofs and an equivalence relation between nodes of the proofs, a new proof can be formed by selecting a *replaced* node in a *target* proof, finding an equivalent *replacing* node in a *contributing* proof (which can be the target proof itself or a different proof), and replacing the sub-DAG rooted at the replaced node in the target proof by the sub-DAG rooted at the replacing node of the contributing proof. Any nodes in the replaced sub-DAG that are used in another part of the target proof are retained, and any duplicated axioms from the problem (leaf formulae of the proof) are merged.

Figure 2 illustrates a combining step based on two original proofs for the simple propositional example introduced in Section 3. The left hand upper proof is the target proof, with the node containing the formula e as the replaced node, so that the outlined sub-DAG is replaced. The right hand upper proof is the contributing proof, with the node containing the formula e as the replacing node (based on syntactic equivalence), so that the outlined sub-DAG replaces the sub-DAG in the target proof. The lower proof is the combined proof. Note that the nodes containing a in the target proof are retained in the combined proof because they are also used in another part of the target proof. Note also that the node containing $e \Leftrightarrow b$ is merged as the parent of the node containing $\neg e \mid b$ from the target proof, and as the parent of the node containing $\neg b \mid e$ in the replacing sub-DAG.

[3] 'C' - Clustering coefficient, 'sw' - scaled weight.

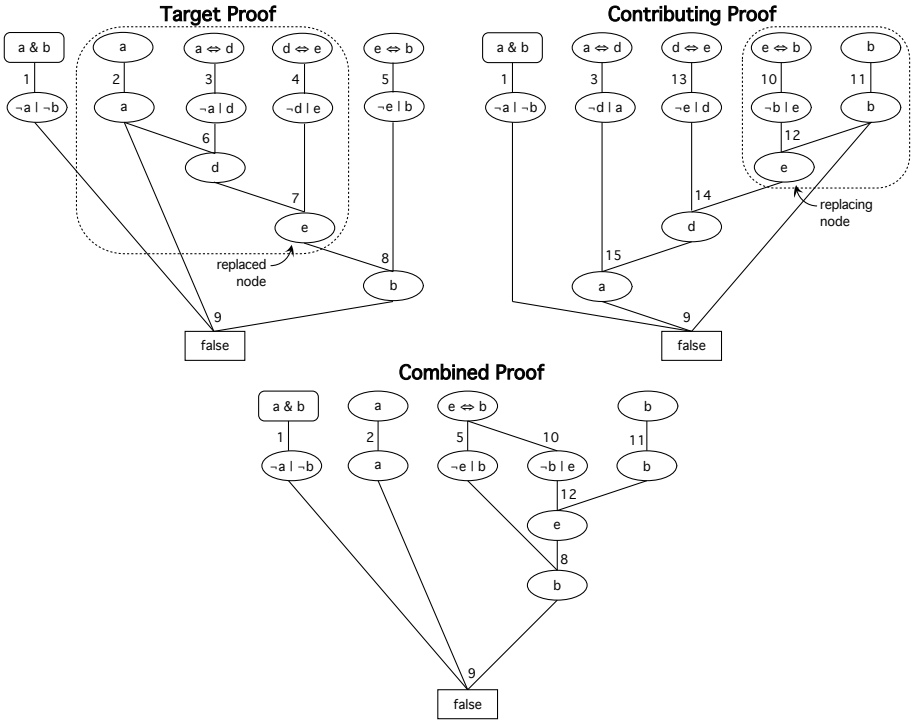

Figure 2: One combining step

The C_{sw} value of Section 3 is used to compare original and combined proofs. Given a set of original proofs, a proof artifact α, and a proof P to be evaluated, $C_{sw}^{\alpha}(P)$ is the C_{sw}^{α} value for the set consisting of the original proofs plus (a distinct copy of) P. Following the example above, using the leaf formulae as the proof artifact of interest, the Jaccard similarity between the target and contributing proofs is 0.66, and between each of the two original proofs and the combined proof it's 0.50. $C_{sw}^{Lf}(Target)$ is 0.7576, and $C_{sw}^{Lf}(Combined)$ is 0.4241. The combined proof is thus (as might be expected) more different than the target proof, from the set of original proofs with respect to leaf sets.

The heuristic of "maximal difference from the original proofs" is the basis for the greedy hill-climbing algorithm that is used to generate a series of new combined proofs based on a set of original proofs. The algorithm is shown in Figure 3. The algorithm gives preference to *self-combining* steps, in which the target proof is also used as the contributing proof, over *non-self-combining* steps that graft in sub-DAGs from an original proof. Self-combining steps serve to "optimize" the target proof, e.g., collapsing sequences of equivalent nodes. The heuristic is used to rank alternative combined proofs, and also to determine if the best combined proof is better than the target proof (in which case the (local) maxima has not yet been reached). The heuristic aims to find non-self-combined proofs that are different (i.e., with smaller C_{sw} values) from the original proofs, first with respect to their leaf sets, then with respect to their inferred formula sets, and lastly with respect to their inferences. When selecting the initial target proof from the set of original proofs, a proof with the largest C_{sw}^{If} is chosen. This preference selects an original proof that has the most formulae that can be used as replaced formulae, hence maximizing the number of possible combined proofs that can be formed. The initial selection also relies on a preference ordering on the ATP systems, which has to be provided by the user.

Figure 4 shows a sequence of combinations as generated by the hill-climbing algorithm, with the two upper proofs of Figure 2 as the original proofs. For this simple illustrative example

```
   /* Choose the first target from the original proofs                    */
 1 BestProofs = the subset of OriginalProofs with maximal C_sw^If;
 2 Target = the proof in BestProofs from the most preferred ATP system;
   /* Start hill-climbing                                                */
 3 repeat
      /* Try combine with itself as the contributing proof               */
 4    CombinedProofs = Combine(Target,{Target});
 5    if CombinedProofs ≠ ∅ then
 6       BestProofs = BestByArtifacts(CombinedProofs);
 7    else
         /* Try combine with original proofs as the contributing proof   */
 8       CombinedProofs = Combine(Target,OriginalProofs);
 9       BestProofs = BestByArtifacts(CombinedProofs ∪ {Target});
10    end
      /* Select the next target proof                                    */
11    if Target ∈ BestProofs then
12       Exit with Target as final proof;
13    else
14       Target = any proof in BestProofs;
15    end
16 until Target == null ;

   /* Function to select the best target proof based on artifacts        */
17 Function BestByArtifacts(Proofs) begin
      /* Prefer proofs with different axioms from the original proofs    */
18    BestProofs = the subset of Proofs with minimal C_sw^Lf;
      /* Next prefer proofs with the same/different inferred formulae as the
         original proofs (see explanation in text)                       */
19    BestProofs = the subset of BestProofs with minimal C_sw^If;
      /* Finally prefer proofs with different inference steps            */
20    BestProofs = the subset of BestProofs with minimal C_sw^Is;
21    return BestProofs;
22 end
```

Figure 3: Hill-climbing combining proofs

syntactic equivalence is adequate for identifying replaced and replacing nodes (see Section 5 for stronger notions of equivalence). The original proof A is selected as the initial target. The first iteration combines Original Proof A with itself as the contributing proof, collapsing the sequence of two nodes containing the formula a, to produce Combined Proof 1. The second iteration continues with Combined Proof 1 as the target proof and Original proof B as the contributing proof. The sub-DAG root at the node containing b is replaced, to produce Combined Proof 2. Another iteration does not produce any more different proofs, and the hill-climbing stops. Table 1 shows the C_{sw} values for the four proofs.

The implementation of the combining process takes advantage of PML's ability to store multiple justifications (PML InferenceSteps) for a formula in a proof (a PML NodeSet), as explained in Section 2. For each node in the target proof, all equivalent ancestor nodes in the target proof, and all equivalent nodes in the original proofs, are identified. The inferences

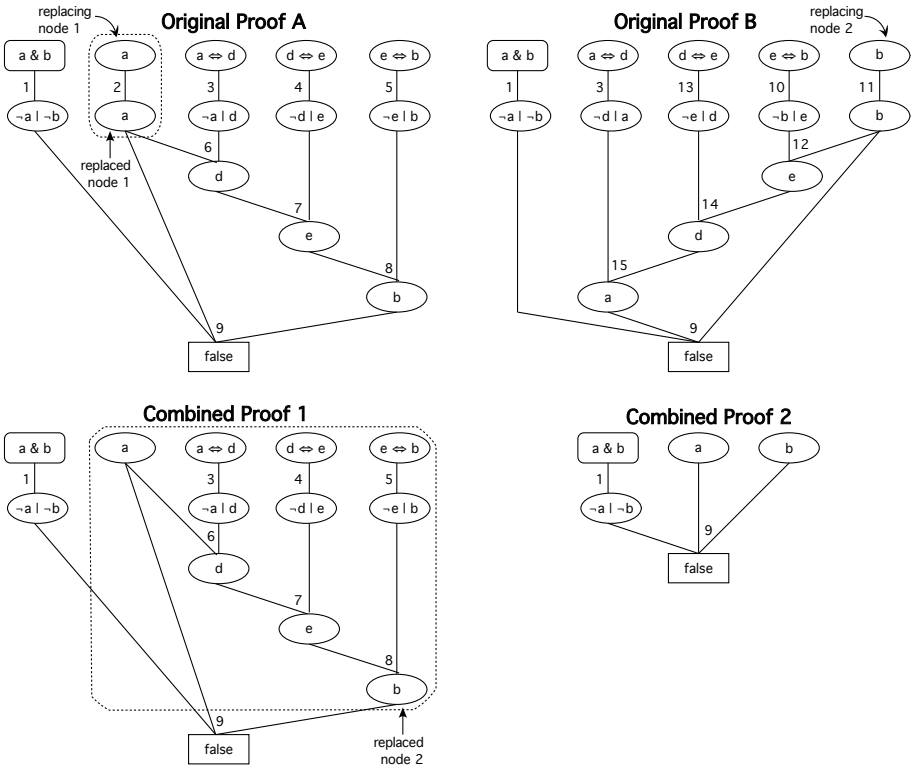

Figure 4: A sequence of combining steps

Proof	C_{sw}^{Lf}	C_{sw}^{If}	C_{sw}^{Is}
Original A	1.0000	0.8483	0.5616
Original B	1.0000	0.8483	0.5616
Combined 1	1.0000	0.8736	0.5616
Combined 2	0.6250	0.6405	0.4235

Table 1: C_{sw} values for sequence of combining steps

that produce these equivalent nodes are added as alternative justifications for the node in the target proof. In this way all the possible justifications for all the nodes in the target proof are captured in one PML structure. Each combined proof is then extracted by selecting a different inference for a node (the replaced node) in the target proof. Note that if the different inference came from the target proof itself, then a self-combining step has been implemented, otherwise a non-self-combining step has been implemented.

The greedy nature of the hill-climbing means that the process does not necessarily find the most different possible combined proof, i.e., the process is not "complete". A broader search, e.g., a beam search or best-first search, might produce more different proofs. A full search of the space of combining possibilities might be necessary to find the most different proof. This is a topic for further investigation.

5 Example Use

The proof combining process has been tested on a range of problems and their proofs. The problems are all in full first-order logic, and the proofs were produced by the ATP systems EP 1.2 [20], Metis 2.3 [9], SInE 0.4 [8], and Vampire 0.6 [17]. In each case the proofs from the same problem were combined, i.e., all the proofs had axioms from the problem's axiom set. In the cases where different problems use the same axiom set it is possible to combine proofs for different problems, but this has been left for future work. As the problems and solutions are in full first-order logic, a simple syntactic notion of equivalence is inadequate. For example, under syntactic equivalence the commutativity, associativity, and reversibility of logical operators would result in "obviously equivalent" formulae not being recognized as such, e.g., $\forall X(q(X) \Leftarrow p(X))$ would not be recognized as equivalent to $\forall X(p(X) \Rightarrow q(X))$. More subtlely, as the ATP systems all convert the problems to clause normal form, Skolem function symbols are typically introduced, and different ATP systems use different naming conventions for these symbols. Therefore a more general notion of formula equivalence was implemented, taking into account the properties of logical operators, possible reordering of quantifications, and naming of Skolem symbols.

As an exemplar use of proof combining, a simplification of the general education rules for obtaining a Bachelor of Science degree from the College of Arts and Sciences at the University of Miami[4] was encoded. The rules require that students complete requirements in English composition, Humanities, Natural science, Mathematics, Social science, a Second language, and Writing. In most of these areas there are alternative ways of completing the requirements. The encoding is provided in Appendix A. A proof of the conjecture that "it is possible to complete the general education requirements" finds one possible combination of the alternatives. By combining a few different proofs, new combinations of the alternatives are found. The combining process used proofs by EP and Metis. As noted in Section 4, the algorithm relies on a preference ordering on the ATP systems' – for these experiments the order chosen was EP then Metis.[5]

Three original proofs, two by EP and one by Metis, were used. The combining process chose one of the EP proofs as the initial target. The proof DAG is shown in the left hand side of Figure 5, as rendered by IDV [25]. An examination of the axioms used shows that EP chose Art (Humanities), Biology (Natural science), Computer Science (Mathematics), Anthropology (Social science), Arabic (Second language), and Philosophy (Writing). Philosophy could have been used to also satisfy the Humanities requirement, but EP chose to take the extra Art course. The combining process first does 13 self-combining steps, which collapse sequences of equivalent formulae in EP's proof. Following that there are five non-self-combining steps interleaved with 28 self-combining steps. The five non-self-combining steps replace sub-DAGs representing alternative ways of satisfying the various general education requirements. The 28 self-combining steps collapse sequences of equivalent formulae in the replacing sub-DAGs from the non-self-combining steps.

The right hand side of Figure 5 shows the final combined proof. An examination of the axioms used shows that the combined proof chose Art History (Humanities, from Metis' proof), Biology (Natural science, from the initial EP proof), Statistics (Mathematics, from the other EP proof), Psychology (Social science, from the other EP proof), Japanese (Second language, from the other EP proof), and a specialized writing course (Writing, from the other EP proof). The first of the five non-self-combining steps makes a comprehensive change to the EP proof, replacing the sub-DAG rooted at the second-to-deepest node in the target proof by the corresponding sub-DAG from the other EP proof - this step effectively replaced the original EP proof by the

[4] `http://www.miami.edu/bulletins`

[5] This ordering was only weakly motivated, based on experience with the systems.

other EP proof. The second non-self-combining step replaces the sub-DAG for the Humanities with that from Metis' proof. The third non-self-combining step replaces the sub-DAG for the Science with that from the original EP proof. All those three steps reduce the C_{sw}^{Lf} value. The last two non-self-combining steps replace parts of the proof in which there is no choice about the education requirements, but using different sequences of inferences that produce different C_{sw}^{If} values.

Table 2 shows the C_{sw} values for the three original proofs, the target proof after the 13 self-combining steps, and the final combined proof. All the heuristic values are smaller in the final combined proof. The greedy nature of the hill-climbing algorithm leads to a proof (and hence the way of completing the general education requirements) that is different from the original three, according to the measures described in Section 3.

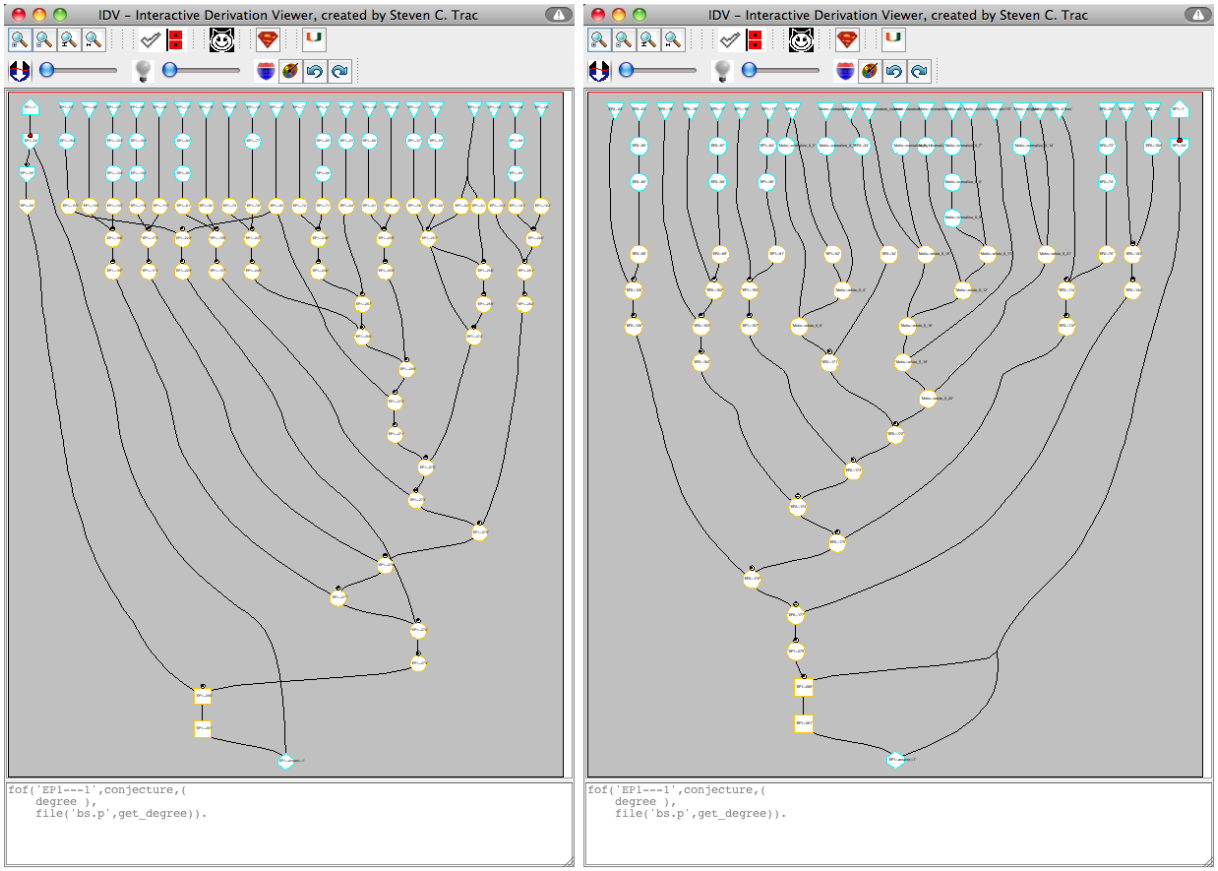

Figure 5: The initial and final proofs

6 Conclusion

This paper has described a process for combining proofs to produce new different proofs. The process uses measures of proof difference based on the Jaccard similarities between sets of proof artifacts (leaf formulae, inferred formulae, and inference steps), and implements a hill-climbing approach that generates proofs that are successively more different from the original proofs. An example has shown how this process can be applied in a practical application.

Proof	C_{sw}^{Lf}	C_{sw}^{If}	C_{sw}^{Is}
EP chosen	0.6466	0.7105	0.6155
EP other	0.6274	0.7045	0.6475
Metis	0.6466	0.6475	0.5166
After 13	0.6466	0.7063	0.5791
Combined	0.6270	0.6815	0.5428

Table 2: C_{sw} values for the original and combined proofs

There are four items of work planned for the immediate future. The first is to do more extensive testing, to confirm that the process is useful over a broad range of problem domains, and exploit the full potential of proof combining. This testing will use problems from the TPTP and proofs from the TSTP. The second is to combine proofs that come from different problems that use the same axiom set, e.g., a suite of problems that are based on a common axiomatization of set theory. The third is to try broader search algorithms, to produce proofs that are more different from the original proofs, and hence regain some of the completeness lost by the greedy hill-climbing. The fourth is to further generalize the notion of formula equivalence, e.g., using ATP to prove that two formulae are equivalent (modulo different Skolem symbol names), or to prove that the formula in the replacing node implies the formula in the replaced node.

In the longer term the proof combining process will be applied to a broad range of proof-like structures, with a range of motivations for producing combined proofs. One motivation comes from settings where there are multiple sources of information, and some sources may be less certain than others. When proofs can be combined and the same conclusions reached from alternative sources, trust may be increased. Identifying different derivations for the same conclusion may be of value when considering explanation strategies. Sometimes one derivation to a conclusion may be more useful for computation and another derivation may be more useful for presenting explanations. For example, a refutation style approach may be useful for computation while a more direct forward chaining style approach may be more appropriate for an explanation. Some derivations may be best for some users while others may be best for different users. For example, some users may prefer derivations that depend more heavily on certain reasoning tools.

A particular type of "proof-like structure" to which the proof combining process can be applied is scientific provenance traces. It is important to note two aspects of scientific activities to understand the connection between these distinct areas of endeavor:

- It is typical for scientists to document the provenance of their scientific products, i.e., the way the scientists collect and process data to derive their scientific products, so that they can claim the reproducibility of their scientific activities.
- The provenance of scientific products and underlying activities is often a description of derivation traces that are simply less formal than proofs in logic. Proofs are a special case of provenance information, where the conclusion of each step of a derivation is a formal sentence (as opposed to a generic piece of information), and inference rules are often defined as patterns over these sentences.

With this understanding, the proof combining process should be transferable to scientific provenance. For example, it is common for geoscientists to combine multiple types of evidence to get an idea of the subterranean features that exist within some geographical region. For gravity data about the earth, geoscientists are often concerned only with anomalies in the data, which often indicate the presence of a water table or oil reserve. In a contour map, such anomalies are illustrated as a set of contour lines with very close proximity, indicating a drastic change

in gravity (or whatever data is being mapped). However, these anomalies have the potential to be artificial – simply imperfections introduced during the map generation process. The use of seismograms can produce a higher resolution three-dimensional tomography of the same subterranean features identified by gravity contour maps. If the contour map and a 3D tomography of a given region are compared, it should be possible to determine if the anomalies are artificial imperfections or features to be further investigated. If the provenance of these scientific products are further compared, it may even be possible to identify the cause of the imperfection. However, without the use of automated "proof" combining techniques, these comparisons have to be manually identified and enabled by geoscientists who understand the interdependencies between data. Common aspects of provenance traces such as "subterranean features" and "regions of interest" may allow the proof combining process to be applied.

References

[1] S. Autexier, C. Benzmüller, D. Dietrich, A. Meier, and C-P. Wirth. A Generic Modular Data Structure for Proof Attempts Alternating on Ideas and Granularity. In M. Kohlhase, editor, *Proceedings of the 4th International Conference on Mathematical Knowledge Management*, number 3863 in Lecture Notes in Artificial Intelligence, pages 126–142. Springer-Verlag, 2006.

[2] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-Time Reductions of Resolution Proofs. In H. Chockler and A. Hu, editors, *Proceedings of the 4th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, number 5394 in Lecture Notes in Computer Science, pages 114–128. Springer-Verlag, 2009.

[3] L. Cheikhrouhou and V. Sorge. PDS - A Three-Dimensional Data Structure for Proof Plans. In R. Braham and M. Mohammadian, editors, *Proceedings of the International Conference on Artificial and Computational Intelligence*, pages 144–149, 2000.

[4] A. Fiedler. P.rex: An Interactive Proof Explainer. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 416–420. Springer-Verlag, 2001.

[5] P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Compression of Propositional Resolution Proofs via Partial Regularization. In N. Bjorner and V. Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, page To appear. Springer-Verlag, 2011.

[6] J-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[7] S. Hetzl. *Proof Profiles. Characteristic Clause Sets and Proof Transformations*. VDM, 2008.

[8] K. Hoder and A. Voronkov. Sine Qua Non for Large Theory Reasoning. In V. Sofronie-Stokkermans and N. Bjœrner, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, page To appear. Springer-Verlag, 2011.

[9] J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Munoz, editors, *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[10] P. Jaccard. Étude Comparative de la Distribution Florale Dans une Portion des Alpes et du Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[11] M. Kohlhase. *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2006.

[12] W. McCune and O. Shumsky-Matlin. Ivy: A Preprocessor and Proof Checker for First-Order Logic. In M. Kaufmann, P. Manolios, and J. Strother Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, number 4 in Advances in Formal Methods, pages 265–282. Kluwer Academic Publishers, 2000.

[13] D. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium, 2004. World Wide Web Consortium (W3C) Recommendation.

[14] T. Opsahl and P. Panzarasa. Clustering in Weighted Networks. *Social Networks*, 31(2):155–163, 2009.

[15] P. Pinheiro da Silva, D.L. McGuinness, and R. Fikes. A Proof Markup Language for Semantic Web Services. *Information Systems*, 31(4-5):381–395, 2006.

[16] P. Pinheiro da Silva, G. Sutcliffe, C. Chang, L. Ding, N. del Rio, and D. McGuinness. Presenting TSTP Proofs with Inference Web Tools. In R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 4th International Joint Conference on Automated Reasoning*, number 373 in CEUR Workshop Proceedings, pages 81–93, 2008.

[17] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[18] S. Rollini, R. Bruttomesso, and N' Sharygina. An Efficient and Flexible Approach to Resolution Proof Reduction. In S. Barner, I. Harris, D. Kroening, and O. Raz, editors, *Proceedings of the 6th International Conference on Hardware and Software: Verification and Testing*, number 6504 in Lecture Notes in Computer Science, pages 182–196. Springer-Verlag, 2011.

[19] M. Schiller, D. Dietrich, and C. Benzmüller. Proof Step Analysis for Proof Tutoring – a Learning Approach to Granularity. *Teaching Mathematics and Computer Science*, 6(2):325–343, 2008.

[20] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[21] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[22] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[23] G. Sutcliffe, C. Chang, L. Ding, D. McGuinness, and P. Pinheiro da Silva. Different Proofs are Good Proofs. In D. McGuinness, A. Stump, G. Sutcliffe, and C. Tinelli, editors, *Proceedings of the IJCAR 2010 Workshop on Evaluation Methods for Solvers and Quality Metrics for Solutions*, 2010.

[24] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81, 2006.

[25] S. Trac, Y. Puzis, and G. Sutcliffe. An Interactive Derivation Viewer. In S. Autexier and C. Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.

A Appendix

```
fof(get_degree,conjecture,
    degree ).

fof(degree,axiom,
    ( ( composition & humanities & science & math & social_science & language & writing )
   => degree ) ).

fof(composition,axiom,
    ( ( eng105 & eng106 )
   => composition ) ).

fof(composition_courses,axiom,
    ( eng105 & eng106 ) ).

fof(humanities,axiom,
    ( ( art & literature & religion & phi115 )
   => humanities ) ).
```

```
fof(art,axiom,
    ( ( artXXX | arhXXX | danXXX | mcyXXX | thaXXX )
   => art ) ).

fof(artXXX,axiom,artXXX ).  fof(arhXXX,axiom,arhXXX ).  fof(danXXX,axiom,danXXX ).
fof(mcyXXX,axiom,mcyXXX ).  fof(thaXXX,axiom,thaXXX ).

fof(literature,axiom,
    ( eng2XX
   => literature ) ).

fof(literature_courses,axiom,
    ( eng2XX ) ).

fof(religion,axiom,
    ( relXXX
   => religion ) ).

fof(religion_courses,axiom,
    ( relXXX ) ).

fof(phi115,axiom,
    ( phi115 ) ).

fof(science,axiom,
    ( ( bilXXX | chmXXX | ecsXXX | geoXXX | mscXXX | phyXXX )
   => science ) ).

fof(bilXXX,axiom,bilXXX ).  fof(chmXXX,axiom,chmXXX ).  fof(ecsXXX,axiom,ecsXXX ).
fof(geoXXX,axiom,geoXXX ).  fof(mscXXX,axiom,mscXXX ).  fof(phyXXX,axiom,phyXXX ).

fof(math,axiom,
    ( ( mth162 & ( cscXXX | staXXX) )
   => math ) ).

fof(mth162,axiom,mth162 ).  fof(cscXXX,axiom,cscXXX ).  fof(staXXX,axiom,staXXX ).

fof(social_science,axiom,
    ( ( apyXXX | ecoXXX | gegXXX | hisXXX | intXXX | polXXX | psyXXX | socXXX )
   => social_science ) ).

fof(apyXXX,axiom,apyXXX ).  fof(ecoXXX,axiom,ecoXXX ).  fof(gegXXX,axiom,gegXXX ).
fof(hisXXX,axiom,hisXXX ).  fof(intXXX,axiom,intXXX ).  fof(polXXX,axiom,polXXX ).
fof(psyXXX,axiom,psyXXX ).  fof(socXXX,axiom,socXXX ).

fof(language,axiom,
    ( ( arb2XX | chi2XX | fre2XX | ger2XX | gre2XX | heb2XX | ita2XX | jap2XX |
        lat2XX | por2XX | spa2XX )
   => language ) ).

fof(arbXXX,axiom,arb2XX ).  fof(chiXXX,axiom,chi2XX ).  fof(freXXX,axiom,fre2XX ).
fof(gerXXX,axiom,ger2XX ).  fof(greXXX,axiom,gre2XX ).  fof(hebXXX,axiom,heb2XX ).
fof(itaXXX,axiom,ita2XX ).  fof(japXXX,axiom,jap2XX ).  fof(latXXX,axiom,lat2XX ).
fof(porXXX,axiom,por2XX ).  fof(spzXXX,axiom,spa2XX ).

fof(wwwXXX_writing,axiom,
    wwwXXX => writing ).

fof(wwwXXX,axiom,wwwXXX).

fof(hisXXX_writing,axiom,
    hisXXX => writing ).

fof(eng2XX_writing,axiom,
    eng2XX => writing ).

fof(phi115_writing,axiom,
    phi115 => writing ).
```