Common subproofs in proof pairs

Guillermo Morales-Luna [*]

Abstract—**In any formal theory, a proof is a sequence of well formed formulas (wff). Here, we consider the digraph whose nodes are proofs and the edges are pairs of proofs such that the second proof is obtained from the first proof by the application of an inference rule. A path is thus a proof extension, and the corresponding shortest path problem consists of finding the minimum number of inference rules applications in order to extend a proof from another given proof. A slight modification is considered by introducing a probability associated to each inference rule from a current proof. Also a corresponding version of the *Longest Common Subsequence* problem is stated: Given two proofs, find the longest common subproof. The dual problem is to find the *Shortest Common Superproof*. We discuss these problems with respect to the decidability of the given formal theory. Our main purpose is to point similarities among the LCS problem, rather important in areas as Bioinformatics, and the characterization of analogy between formal proofs in first order theories. We do not pretend to improve some other well known methods for constructing proofs based on analogy.**

Keywords: Decidability, relative complexity

1 Introduction

Automatic theorem proving has been developed by several general procedures including *analogy*. In this case, given a *source* theorem and a *target* theorem some similarities are looked for in order to transform the source proof into a proof of the target theorem. One approach is the so called *derivational analogy* [1]. As an alternative approach, since *Curry-Howard Isomorphism* [2] establishes a correspondence between formal logics and computational calculi as a type theory, a methodology has been presented in [3] for proof transformation, by representing the proofs in a term-functional language and the proof transformations as rewriting rules in the language giving a whole impementation using NuPRL [4].

In this paper we do not deal with the processes of transforming a proof into another proof. Rather, we want to compare any two theorems in a formal logic in order to decide how similar are they in terms of corresponding proofs.

The classical *Edit Problem* consists of, given an alphabet Σ and two strings $\sigma, \tau \in \Sigma^*$, to transform one word into the other by using the least number of operations `Insert`, `Replace` and `Delete`, acting on symbols. This last number is the *edit distance* between the strings. The Edit Problem has had a great importance in Bioinformatics [5], it is essential when comparing properties and structures of two strands of RNA, and it is representative of Dynamic Programming paradigms. Bille [6] and Batu [7] have presented extensive treatments for this problem. In [8], a corresponding version for graphs is presented. In [7] an approximation algorithm for the edit distance is given (it decides in sublinear time whether the distance is $O(n^\alpha)$, $0 < \alpha < 1$). The procedures by Vidal *et al.* [9] are already classical examples of efficient computations. The Edit Problem is closely related to the *Longest Common Subsequence* (LCS) of two strings, which is solved in time complexity $O(n/\log n)$. LCS is quite relevant in Bioinformatics as well, and recently [10] it has been used to analyze the evolution of natural languages.

Here we deal with corresponding problems in formal logical theories. Any proof in a theory is indeed a sequence of well formed formulas (wff). Hence, given any two proofs, to find the longest subproof contained in those proofs or the shortest superproof containing the proofs may provide information about the similarities between the given proofs: The closer the longest subproof or the shortest superproof are to the two given proofs, the more similar these proofs are. No Edit Problem can be posed since evidently there is no any possibility of deleting on proofs.

The current methods to solve the string versions of the problems may be translated verbatim to the proof analysis, and all computations will be related to the complexity of the decision problem in the theory.

In the next section we introduce the basic notions and the notation to be used in our presentation. Then we will discus the shortest path problem, and the corresponding versions to LCS and to *Shortest Common Supersequence* (SCS) problems.

2 Preliminaries and notation

2.1 Proofs in formal logics

Let $\mathcal{L}$ be a language for a first order theory. Let $\mathcal{F}$ be the set of well formed formulas and let $\mathcal{A} \subset \mathcal{F}$ be a distin-

[*]Computer Science Department, CINVESTAV-IPN, 07300 Mexico City, Mexico, Tel/Fax: (+52)-555-061-3759. Email: gmorales@cs.cinvestav.mx . This work has been partially supported by Mexican Conacyt.

guished set of axioms. Any *inference rule* is of the form $\frac{F}{\phi}$, where F is a scheme for finite sets of wff's (called the *premises*) and ϕ is a scheme for a wff (called the *consequence*). Let $\mathcal{R}$ be a fixed set of inference rules. A *proof* from a set of hypothesis H is a finite list of wff's $F \in \mathcal{F}^*$ such that for each $i \leq \text{length}(F)$, if ϕ_i is the i-th formula in F, then either it is an axiom, $\phi_i \in \mathcal{A}$, or an element in H, or it is the consequence of an inference rule whose premises have been matched by formulas in F previous to ϕ_i, i. e. for some $\phi_{j_1}, \ldots, \phi_{j_k}$, with indexes $j_1, \ldots, j_k < i$, the instantiated rule $\frac{\phi_{j_1}, \ldots, \phi_{j_k}}{\phi_i}$ matches an inference rule $\frac{F}{\phi}$ in $\mathcal{R}$. If ϕ is the last formula in a proof, then it is said that ϕ is *provable* from H, $H \vdash \phi$. The formal theory $\mathcal{T}(H)$ is the set of all formulas provable from H. The theory $\mathcal{T} = \mathcal{T}(\emptyset)$ consists of all *theorems*. The empty list $\Lambda \in \mathcal{F}^*$ will be considered also as a proof.

The notion of proof depends existentially on the notions for axioms and inference rules. Let us recall a series of classical results: whenever the set of axioms $\mathcal{A}$ and the set of inference rules $\mathcal{R}$ are recursive, then the theory $\mathcal{T}$ is a recursively enumerable set [11]. Thus, if the theory is also complete, then the class of theorems is recursive indeed. Hence, the so called *Łoś-Vaught Test*: "Any theory with no finite models that is categorical is complete", will imply that any such theory is recursive. On the other hand, if the theory is strong enough to weakly represent every recursive property, then the theory cannot be decidable [12]. The First Gödel's Incompleteness Theorem implies, thus, that no completion of arithmetic can be recursively axiomatizable. Clearly, the complexity of any theory is determined by the complexities of its set of axioms and its set of inference rules.

Let $\lceil\cdot\rceil : \mathcal{F} \to \mathbb{N}$ be a computable coding function. Let $F_{\mathcal{A}}$ be a representation function for the set of axioms: $\forall \phi \in \mathcal{F}\ (\phi \in \mathcal{A} \Leftrightarrow F_{\mathcal{A}}(\lceil\phi\rceil) = 0)$; and for each inference rule $R = \frac{\tilde{\phi}_0, \ldots, \tilde{\phi}_{k-1}}{\tilde{\phi}_k} \in \mathcal{R}$ let F_R be a corresponding representation function: for any $\phi_0, \ldots, \phi_{k-1}, \phi_k \in \mathcal{F}$,

$$\begin{aligned} & \frac{\phi_0, \ldots, \phi_{k-1}}{\phi_k} \text{ matches } R \\ \Leftrightarrow \quad & F_R(\lceil\phi_0\rceil, \ldots, \lceil\phi_{k-1}\rceil, \lceil\phi_k\rceil) = 0. \end{aligned}$$

The collection of functions $\mathcal{C} = \{F_{\mathcal{A}}\} \cup \{F_R | R \in \mathcal{R}\}$ is the basic complexity class. Let $\mathcal{D}$ be the class of sets that can be represented with second-order formulas consisting of quantifier-free formulas involving symbols in $\mathcal{C}$ and at most one string of existential quantifiers $\exists f \in \mathcal{C}$ or, in other words, $\mathcal{D}$ consists of the Σ_1 second-order formulas. Naturally, if $\mathcal{C}$ is in turn representable in $\mathcal{T}$ then those second-order representations have equivalent first-order representations. It is direct that the notion of proof can be decided by an oracle in $\mathcal{D}$.

2.2 Permutations of proofs

Let $\mathcal{P}$ be the set of proofs. If we denote by $\mathcal{F}^*$ the collection of finite sequences of elements in $\mathcal{F}$, then $\mathcal{P} \subset \mathcal{F}^*$.

For any proof $F = [\phi_i | 0 \leq i \leq n-1] \in \mathcal{P}$, if its length is $n = \text{length}(F)$, and π is a permutation of the set of indexes $\{0, \ldots, n-1\}$, let us define $\pi(F) = \left[\phi_{\pi(i)} | 0 \leq i \leq n-1\right]$ as the wff list whose elements are those of F permuted according to π. We will say that the permutation π is *consistent with* F if $\pi(F) \in \mathcal{P}$: $\pi(F)$ is also a proof. For any two $F, G \in \mathcal{P}$ let us say $F \equiv_w G$ if there is a permutation consistent with F such that $G = \pi(F)$, or in other words, if G is just a rewriting of F.

Clearly, $\equiv_w$ is an equivalence relation among proofs.

In order to decide whether two proofs are equivalent let us consider the *height* $h_F(\phi)$ of any wff ϕ in a proof F. Namely, if F is a proof and $\phi \in F$ is either an axiom or a hypothesis, then $h_F(\phi) = 0$, and if ϕ is the consequence of a rule with set of premises $\{\phi_0, \ldots, \phi_{k-1}\}$, then $h_F(\phi) = 1 + \max\{h_F(\phi_j) | j = 0, \ldots, k-1\}$. If the wff's in F are permuted according to their height values, then an equivalent proof is obtained, and if at each segment of constant height the wff's are ordered lexicographically then a "canonical" proof is got.

Then two proofs are equivalent according to $\equiv_w$ whenever they produce the same canonical proof. This decision procedure involves at most $O(n^2)$ tests of wff equality, where n is the length of the involved proofs. Indeed, the decision process for predicate $F \equiv_w G$ is representable in $\mathcal{D}$ as well. Let $\mathcal{P}_0 = (\mathcal{P}/\equiv_w)$ be its quotient space. For any $F \in \mathcal{P}$ let us identify F with its equivalence class $[F] \in \mathcal{P}_0$.

2.3 Graph of proofs up to permutations

Let $\mathcal{G}_{\mathcal{T}}$ be the graph whose set of nodes is $\mathcal{P}_0$, and whose edges are of the form (F, G) such that the proof F produces the proof G by the application of some inference rule in $\mathcal{R}$: G is thus a *successor* of F, and F is a *precedent* of G.

For each $F \in \mathcal{P}_0$, let succ(F) be the set of G's such that (F, G) is an edge in the graph $\mathcal{G}_{\mathcal{T}}$. The *out-degree* of F is the cardinality of succ(F). The out degree of a node gives the number of possibilities to proceed to extend proof from the current node. In order to give preferences among these possibilities, let us assume that for each $F \in \mathcal{F}^*$ there is a probability density function Pr_F over succ(F),

$$\begin{aligned} \forall G \in \text{succ}(F): \quad & 0 \leq \text{Pr}_F(G) \leq 1 \text{ and} \\ & \sum \{\text{Pr}_F(G) | G \in \text{succ}(F)\} = 1. \end{aligned}$$

Each edge (F, G) will be labeled by the probability real number $\mathrm{Pr}_F(G)$ and will have associated a *weight*, or *distance*, equal to $-\log(\mathrm{Pr}_F(G))$ (obviously, if $\mathrm{Pr}_F(G) = 0$ then the weight is $+\infty$).

A path from a proof F to a proof G is a procedure to obtain G from some premises involved in F.

Given $F_0 \in \mathcal{P}_0$, let $\mathrm{succ}^{(0)}(F_0) = \{F_0\}$ and, for any integer $n \geq 1$, let

$$\mathrm{succ}^{(n)}(F_0) = \bigcup \left\{ \mathrm{succ}(G) | G \in \mathrm{succ}^{(n-1)}(F_0) \right\}. \qquad (1)$$

$\mathrm{succ}^{(n)}(F_0)$ is thus the collection of all proofs that extend F_0 by applying a sequence of exactly n inference rules. Any $F_n \in \mathrm{succ}^{(n)}(F_0)$ is connected to F_0 by a sequence on n edges $(F_0, F_1), (F_1, F_2), \ldots, (F_{n-1}, F_n)$. Let us label this path by the product $\mathrm{Pr}^{(n)}_{F_0}(F_n) = \prod_{i=1}^n \mathrm{Pr}_{F_{i-1}}(F_i)$, and consequently its distance is

$$-\log\left(\mathrm{Pr}^{(n)}_{F_0}(F_n)\right) = \sum_{i=1}^{n} \left(-\log\left(\mathrm{Pr}_{F_{i-1}}(F_i)\right)\right). \qquad (2)$$

Clearly, $\mathrm{Pr}^{(n)}_{F_0}$ determines a probability density over $\mathrm{succ}^{(n)}(F_0)$. Besides, the search of the path with highest probability from a proof F to a proof G, coincides with the search of the shortest path connecting those proofs, if any, in $\mathcal{G}_{\mathcal{T}}$.

Dijsktra's algorithm arises naturally as a very first candidate to calculate shortest paths.

On the other hand, for any two proofs $F, G \in \mathcal{P}_0$ let us define $F \preceq G$ if and only if either $F = G$ or there is a path in $\mathcal{G}_{\mathcal{T}}$ connecting F with G. $\preceq$ is an ordering in $\mathcal{P}_0$. For any two proofs F, G, its intersection $F \cap G$ is a lower bound of the pair $\{F, G\}$ and its juxtaposition $F * G$ (with suppressed repetitions of formulas) is an upper bound. Since the proofs are of finite length we have that $(\mathcal{P}_0, \preceq)$ is a lattice and its diagram is precisely the graph $\mathcal{G}_{\mathcal{T}}$. Hence the following problems can be posed in $\mathcal{G}_{\mathcal{T}}$:

Shortest path.
If F and G are two proofs such that $F \preceq G$, calculate the most probable path connecting F with G.

Longest Common Subsequence (LCS).
Calculate $F \wedge_{\mathcal{T}} G = \mathrm{Inf}\{F, G\}$, for any given two proofs $F, G \in \mathcal{P}_0$.

Shortest Common Supersequence (SCS).
Calculate $F \vee_{\mathcal{T}} G = \mathrm{Sup}\{F, G\}$, for any given two proofs $F, G \in \mathcal{P}_0$, .

3 Shortest path

Given two proofs F and G, a translation of Dijkstra's algorithm to compute the shortest distance $\rho_F(G)$ between F and G in the graph $\mathcal{G}_{\mathcal{T}}$ is the following:

```
1. Initially, let $\mathcal{W} := \{F\}$ and
   $\mathcal{W}_1 := \{F_1 \in \mathrm{succ}(F) | F_1 \preceq G\}$.
2. For each $F_1 \in \mathcal{W}_1$ do $\rho_F(F_1) := -\log(\mathrm{Pr}_F(F_1))$.
3. While $G \notin \mathcal{W}$ do
   (a) let $F_0 := \mathrm{ArgMin}\{\rho_F(F_1) | F_1 \in \mathcal{W}_1\}$;
   (b) let $\mathcal{W}_2 := \{F_1 \in \mathrm{succ}(F_0) - \mathcal{W}_1 | F_1 \preceq G\}$;
   (c) for each $F_1 \in \mathcal{W}_2$ do
       $\rho_F(F_1) := \rho_F(F_0) - \log(\mathrm{Pr}_{F_0}(F_1))$;
   (d) for each $F_1 \in \mathcal{W}_1$ do assign to $\rho_F(F_1)$ the value
       $\min\{\rho_F(F_1), \rho_F(F_0) - \log(\mathrm{Pr}_{F_0}(F_1))\}$;
   (e) $\mathcal{W} := \mathcal{W} \cup \{F_0\}$;
   (f) suppress $F_0$ from $\mathcal{W}_1$;
   (g) $\mathcal{W}_1 := \mathcal{W}_1 \cup \mathcal{W}_2$
4. Output $\rho_F(G)$
```

In the above pseudocode, the operator ArgMin at the right side of assignment 3.(a) denotes the argument F_1 that makes the map ρ_F to attain its mimimum value over the class $\mathcal{W}_1$.

Given two proofs F, G, with $F \preceq G$, departing from $F_0 = F$, and advancing through edges in $\mathcal{G}_{\mathcal{T}}$ we look for the shortest path from F to G. Since at each step the current subproof F_0 must be a subsequence of the goal proof G, the algorithm should perform $O(n^2)$ queries, with $n = \mathrm{length}(G)$, about whether a formula in G is the consequence of an inference rule with hypothesis contained in the current subproof F_0. Each query depends on the number of schemes for the inference rules and the unification process. Thus the calculation of shortest paths has time complexity $O(nru(k))$ where r is the number of inference rules schemes, k is the maximum length of formulas in G and u is the time complexity of the unification process in $\mathcal{F}$.

4 Common subsequences

4.1 LCS

For any proof $F \in \mathcal{P}_0$, let us distinguish its last formula as $\Theta(F) = \phi$ and let $\Xi(F) = F - [\phi]$, or equivalently, let us split the proof as $F = \Xi(F) * [\Theta(F)]$.

The following remark serves as basis for the Dynamic Programming approach to find the longest common sequence: For any $F, G \in \mathcal{P}_0$, let

$$F \wedge_{\mathcal{T}} G = (\Xi(F) \wedge_{\mathcal{T}} \Xi(G)) * [\Theta(F)]$$

whenever $\Theta(F) = \Theta(G)$, or let

$$F \wedge_{\mathcal{T}} G = \mathrm{longest\_of}\left\{(\Xi(F) \wedge_{\mathcal{T}} G), F \wedge_{\mathcal{T}} \Xi(G)\right\}$$

otherwise. This gives an algorithm of order $O(\mathrm{length}(F)\,\mathrm{length}(G))$, i.e. proprtional to the product of the lengths of F and G, with respect to tests for equality in $\mathcal{P}_0$. Each such test can be performed in linear time with respect to $\max(\text{length}\,(F), \text{length}\,(G))$ using oracles in the set class $\mathcal{D}$ as defined at the end of section 2.1.

A variant for this problem is to find the common subsequence $H \preceq F, G$ of both sequences F and G, of length $k \leq \min(\text{length}\,(F), \text{length}\,(G))$, with the highest probability among the k-length common subsequences of F and G:

$$H_k(F,G) = \underset{(F_0,H)}{\text{ArgMax}} \left\{ \text{Pr}_{F_0}^{(k)}(H) | \; H \preceq F, G \;\&\; \text{length}\,(H) = k \right\}.$$

Clearly, for any $k \geq 1$, if $H_k(F,G) = (F_0, H)$ then

$$\text{Pr}_{F_0}^{(k)}(H) = \max \left\{ \text{Pr}_{F_0}^{(k-1)}(H_{k-1}(F, F_0)) \cdot \text{Pr}_{F_0}(F_1) | \; F_1 \in \text{succ}(F_0) \;\&\; F_1 \preceq F, G \right\}. \quad (3)$$

According to a Dynamic Programming approach, eq. (3) solves the problem variant also in polynomial time with respect to $\max(\text{length}\,(F), \text{length}\,(G))$ using oracles in the set class $\mathcal{D}$.

4.2 SCS

This problem is dual to LCS. Namely, for any given proof $F \in \mathcal{P}_0$, let us distinguish its first formula as $\text{car}(F) = \phi$ and let $\text{cdr}(F) = F - [\phi]$; thus the proof can be split as $F = [\text{car}(F)] * \text{cdr}(F)$.

For any $F, G \in \mathcal{P}_0$, let

$$F \vee_{\mathcal{T}} G = [\text{car}(F)] * (\text{cdr}(F) \vee_{\mathcal{T}} \text{cdr}(G))$$

whenever $\text{car}(F) = \text{car}(G)$, or let

$$F \vee_{\mathcal{T}} G = \text{shortest\_of} \quad \{\text{car}(F) * (\text{cdr}(F) \vee_{\mathcal{T}} G), \; \text{car}(G) * (F \vee_{\mathcal{T}} \text{cdr}(G))\}$$

otherwise. This gives also an algorithm of order $O\,(\text{length}\,(F)\,\text{length}\,(G))$ w.r.t. equality tests in $\mathcal{P}_0$.

5 Conclusions

Both problems SCS and LCS may be used to decide similarities and closeness among formal proofs, and they may be treated by standard Dynamic Programming methods. However the primitive actions in these procedures consists of provability decisions in the formal theories.

Decidability problems in formal theories are representative of the complexities of those theories. The stated difficult combinatorial problems may relativize polynomially their time complexities to those of decidability.

As a line of future development and practical use of the methods exposed here we want to consider the Curry-Howard Isomorphism [2] to exploit the typed λ-calculus in order to compute more efficiently the distance among two proofs using the metric in eq. (2).

References

[1] E. Melis and J. Whittle, "Analogy as a control strategy in theorem proving," in *Proceedings of the 10th Florida International AI Conference, FLAIRS-97.* also published as DAI Research Paper 840, University of Edinburgh, 1997.

[2] M. H. Sørensen and P. Urzyczyn, *Lectures on the Curry-Howard Isomorphism*, ser. Studies in Logic and the Foundations of Mathematics. Elsevier, 2006, no. 149.

[3] T. B. de la Tour and C. Kreitz, "Building proofs by analogy via the Curry-Howard isomorphism," in *Conference on Logic Programming and Automated Reasoning, LPAR'92.* LNAI 624, Springer, 1992, pp. 202–213.

[4] R. L. Constable and et al, *Implementing Mathematics with the NuPRL proof development system.* Prentice Hall, 1986.

[5] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* CUP, 1997.

[6] P. Bille, *Tree Edit Distance, Alignment Distance and Inclusion*, ser. Technical Report Series. IT University of Copenhagen, 2003, no. TR-2003-23.

[7] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in *Proceedings of the Thirty-fifth ACM Symposium on Theory of Computing, STOC'03.* ACM, 2003, pp. 316 – 324.

[8] E. R. H. A. Robles-Kelly, "Edit distance from graph spectra," in *Proceedings of the Ninth IEEE International Conference on Computer Vision, ICCV-2003.* IEEE, 2003.

[9] E. Vidal, A. Marzal, and P. Aibar, "Fast computation of normalized edit distances," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 17, no. 9, pp. 899–902, 1995.

[10] R. D. Gray and Q. D. Atkinson, "Language-tree divergence times support the anatolian theory of indo-european origin," *Nature*, vol. 426, no. 11, pp. 435–439, 2003.

[11] H. B. Enderton, "Elements of recursion theory," in *Barwise, J. (ed), Handbook of Mathematical Logic.* North Holland, 1977, pp. 527–566.

[12] J. L. Bell and M. Machover, *A Course in Mathematical Logic.* North Holland, 1977.