Abstracting Web Agent Proofs into Human-Level Justifications

Vasco Furtado[1], Paulo Pinheiro da Silva[2], Deborah McGuinness[3], Priyendra Deshwal[4], Dhyanesh Narayanan[3], Juliana Carvalho[1], Vladia Pinheiro[1], CynthiaChang[3]

[1] Universidade de Fortaleza, Fortaleza, CE, Brazil on sabbatical at Stanford University
[2]University of Texas, El Paso, TX, USA
[3] KSL, Stanford University, Stanford, CA, USA
[4]Google Inc., Palo Alto, CA, USA

vasco@ksl.stanford.edu, paulo@utep.edu, dlm@ksl.stanford.edu,pryendra@gmail.com, dhyanesh@ksl.stanford.edu, julianacarvalho@fortalnet.com.br, vladiacelia@terra.com.br, changcs1@stanford.edu

Abstract

Information supporting answer explanations are derived from proofs. One of the difficulties for humans to understand web agent proofs is that the proofs are typically described at the machine-level. In this paper, we introduce a novel and generic approach for abstracting machine-level portable proofs into human-level justifications. This abstraction facilitates generating explanations from proofs on the web. Our approach consists of creating a repository of proof templates, called abstraction patterns, describing how machine-level inference rules and axioms in proofs can be replaced by rules that are more meaningful for humans. Intermediate results supporting machine-level proofs may also be dropped during the abstraction process. The Inference Web Abstractor algorithm has been developed with the goal of matching the abstraction patterns in the repository against the original proof and applying a set of strategies to abstract the proof thereby simplifying its presentation. The tools used for creating and applying abstraction patterns are shown along with an intelligence analysis example.[1]

Introduction

Users should be able to use intelligent web agents to answer complex queries. A "trace" or proof for the agent's final result can be viewed as a combined log describing the information manipulation steps used by services to derive the final result. When a human user requests an explanation of what has been done or what services have been called, the agent can use an explanation component to analyze the trace and generate an explanation.

The ability of software agents to present and justify their reasoning to human users has been studied in the context of knowledge-based systems since the 1970's. The explanations in those systems typically focused on some (understandable) presentation of a reasoning trace [1],[2]. Another generation of explanation systems was introduced with the Explainable Expert System [16] with the goal of designing systems with explanation in mind. These systems, however, typically have the same strong assumptions: data and rules are reliable; and the conceptual domain model is generated by a knowledge engineer along with a domain expert. Another focus area for explanation research is that of explaining answers from theorem proving systems. Recognizing that machine-level proofs are difficult to read by humans, researchers in that context try to automatically transform machine-generated proofs into natural deduction proofs [13], or into assertion-level proofs [7]. The assertion level approach allows for human-level macro steps justified by the application of theorems, lemmas or definitions which are called assertions and that are supposed to be at the same level of abstraction as if the proof was generated by a person. However, the simplifications produced by these approaches are not abstracted enough to be used as explanations because even with rule rewriting the semantics behind the logical formulations are difficult for human users to comprehend.

The distributed and evolving nature and diversity of the web have broken the assumptions underlying previous approaches. For example, information extraction techniques used in an answer derivation process may be unknown in advance and domain knowledge may be acquired from different sources, each source representing knowledge at different levels of granularity. Proofs for answers in these settings may be complex and may contain a large number of inference steps that are inappropriate to show to human users, either because they are obvious or contain too fine a level of granularity

We present a novel approach to abstract machine-level proofs from an agent's reasoning trace into human-level justifications that can better support explanation generation processes. Our abstraction approach addresses both required understandability issues as well as the distributed nature of the web. We believe explanations can be improved by choosing appropriate abstraction levels for presenting proofs including the removal of irrelevant details. Further, humans familiar with the system may have insight into abstraction granularity and can distinguish between relevant/irrelevant details. We provide support for describing how answer justifications represented in machine-level proofs can be abstracted into human-level proofs called *explanations*. The abstraction methods

[1]. This work was partially supported by the Defense Advanced Research Agency (DARPA) through contract #55-300000680 to-2 R2, NSF award #0427275, and DTO contract #2003*H278000*000.

attempt to be generic and modular by using proof fragment templates called abstraction patterns (APs) that may be used in different proofs. The method is based on these APs that are likely to match fragments of complete proofs. Thus the templates can be reused and even combined in different situations. Using the human-defined APs, the abstraction method is used to rewrite the ground proof by dropping logical (granular) axioms and by introducing a new higher level inference rule encoded in an AP, which is aimed at human-level justifications.

We create an AP repository that can be either generic or domain-specific allowing users to generate customized explanations. An algorithm is used to match the abstraction patterns with the original proof. In order to use the AP repository in distributed and diverse contexts on the web, we have developed our approach within the Inference Web (IW) framework [10]. In this framework, proofs are encoded in the Proof Markup Language (PML), which works as an Interlingua for Web agent's answer justifications. The IW Abstractor is a new tool realizing the abstraction method..

The Inference Web Framework

Inference Web is a framework for explaining reasoning tasks on the web by storing, exchanging, combining, abstracting, annotating, comparing and rendering proofs and proof fragments provided by question answering applications including Web agents. IW tools provide support for portable and distributed proofs and proof presentation, knowledge provenance, and explanation generation. For example, the IW browser is used to support navigation and presentations of proofs and their explanations. IW data includes proofs and explanations published locally or on the web and represented in PML [10]. PML is built on top of the Ontology Web Language (OWL) and inference step and node set are the two main building blocks of the language. In PML, a proof is defined as a tree of node sets connected through inference steps explaining answer derivation processes. Each node set includes a statement and one or more inference steps. An inference step is a single application of an inference rule over the inference step's antecedents. Inference rules are used to derive node set conclusions (e.g., a well formed formula) from any number of antecedents. An inference step contains pointers to node sets representing antecedent proofs, the inference rule used, and any variable bindings used. A node set is a leaf node in a proof if associated inference steps have no antecedents. Typically, leaf node inference steps are the result of application of either the *Direct Assertion* or *Assumption* rules.

Abstraction Patterns

Abstraction can be formalized as a pair of formal system languages plus a mapping between the languages [6]. IW Abstractor aims to build mappings between machine level and human level proofs, both represented in PML but based on different inference rules and axioms. Each node in a proof represents a set of well formed formulae (wffs) with at least one associated inference step. Given two languages *Lg* and *Lp*, *abs* : *Lg* → *Lp* is an abstraction. *Lg* is the *ground proof*, while Lp is the *abstract proof*. *abs* is the mapping function. An AP is defined by a human expert and matched against the ground language to produce the abstraction. An AP is a meta-proof also encoded in PML where node sets typically represent sentences including meta-variables that will be unified during the matching process. IW Abstractor is specifically interested in using abstraction patterns that result in simpler, more understandable proofs. The *Abstractor* algorithm (see next section) uses the patterns to prune details that inhibit proof comprehension and thus can be used to enable better understanding.

The application of abstraction patterns in a ground proof will produce abstracted proofs (APr). The abstracted proof contains a portion of the ground proof including less detail along with the new inference rule identifying the abstraction pattern applied.

Creating Abstraction Patterns

There is an element of design involved in the process of creating APs and a precise algorithm may not be prescribed for this task. The general steps below provide guidance about the goals of writing abstract patterns:

- Hiding machine-level inference rules, e.g., resolution and universal quantifier elimination.
- Hiding complex axioms that are implicitly identified in the name of rules in APrs.
- Hiding parts of the proof that may be irrelevant (or too obvious) for certain kinds of explanations.
- Removing intermediate results that are unnecessary for human understanding of the justification.

For instance, Transitivity is a property of the *subclass-of* relation as well as many others such as *part-of*, *before*, etc. Many reasoners use axioms describing such a property during answer derivation processes. In doing so, proofs become large and full of detailed steps which may not be appropriate for human-consumption justifications. So, in this context, an AP has the goal of abstracting away steps used by the reasoner to conclude anything based on transitivity.

Editing and Creating Abstraction patterns

One of the ways to construct APs is by modifying existing proofs. An AP editor was developed to help the creation of syntactically correct APs from existing proofs. Figure 1 shows an example of this editing process using the property of transitivity. The example is from the KSL Wine Agent that uses deductive reasoning to match foods and wines. In this case it is matching a wine to Tony's Speciality. After learning that the food should be paired with a white wine, someone has asked for the type of the food being matched. The proof generated by the JTP

reasoner is shown by the IW browser. Tony's specialty turns out to be a crab dish and the reasoner used a number of steps to derive that crab is a type of seafood. This proof fragment is outlined in Figure 1. Using the editor, the AP designer decides to reuse the proof fragment from the nodeset that defines *subClassof* CRAB SEAFOOD.

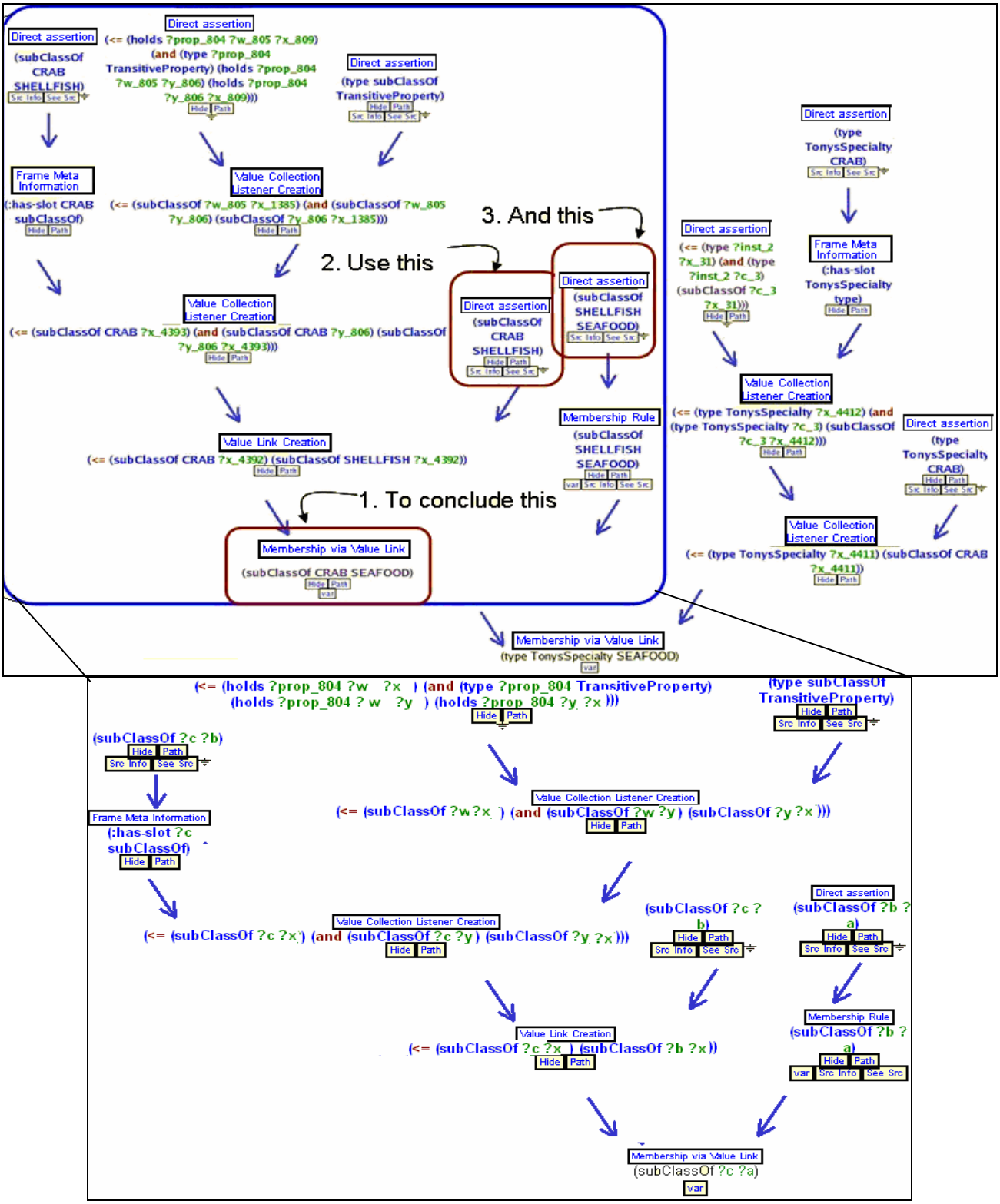

Figure 1 Class transitivity AP from the Seafood domain

S/He drops the nodesets of the right part of the proof tree as well as those that are duplicated because they won't be used in the abstraction.

In order to create its final specification, the designer needs to substitute the identifiers in the ground proof fragment with variables. The AP Editor offers support for this task by means of a global Search/Replace mechanism. For instance replacing subClassOf CRAB SHELLFISH *by ?subClassOf ?c ?b.* The final AP template is depicted in the small window of Figure 1. The basic idea behind such an editing process is that in order to explain that a conclusion was obtained by transitivity only the outlined predicates are necessary from a human end-user point of view. Then, in the process of abstraction when such a pattern is found all the other nodes can be abstracted away.

The Abstractor Algorithm

The IW Abstractor is a tool available as a web service that uses the AP repository to transform IW proofs. It consists of two phases: AP matching and ground proof abstraction. For each AP in the repository, the Abstractor tries initially to match its conclusion then the leaf nodes with those of the ground proof. After a match is found, the pattern is applied to abstract the ground proof by dropping those ground axioms that are subsumed into the abstraction pattern and by replacing the name of the inference rule as a justification for the nodeset conclusion. The abstraction algorithm is described in Frame 1. The main function *abstractProof* calls the match and *abstractNodeset* functions. The match function then calls the *matchLeafNodes* function. We have implemented a unification algorithm based on [14] with linear time complexity. The overall complexity however of the abstractor algorithm is exponential because, maintaining axiom order independence requires permutation of the abstraction patterns' leaf nodes for each unification. Since the typical number of leaf nodes of an AP is small (varying from three to seven), abstraction in practice remains viable. Moreover, we have computed a hash code that is used as an index to guide the pattern-matching phase of the algorithm. The code is built based on the predicate of the nodeset and its variables. Each nodeset keeps a list of hash codes of its antecedents. When the algorithm tries to match an AP's nodeset against a proof's nodeset, the hash code is generated. With this code, the algorithm checks the list of hash codes from the nodesets in the proof and finds out if the code of the template it is looking for is part of the proof. For example, if the algorithm is looking for a match for a nodeset with a template "p x1 x2", where p is the predicate and x1 and x2 are the variables, it generates the hash code for this template and then checks if this hash code is present in the hash code list of the nodesets in the proof. This technique prevents the algorithm from searching the whole proof tree for a nodeset in which the template is not found in the proof.

```
n0: conclusion nodeset of the proof to be
abstracted
abstractProof(n0):
    for each abstraction pattern r,
        if(match(n0, r)<>[])
            abstractNodeset(match(n0, r), n0, r);
        else
            next r;
            if all r's exhausted,
                for all antecedents n of n0,
                    abstractProof(n);
abstractNodeset(L, n, r):
// Verify if the abstraction can be done by
checking if there will be isolated nodes after
abstraction
    If (IsThereIsolatedNodes(L,n) == false)
         Create new nodeset n'
         n'.conclusion = n.conclusion
         add inference step with rule r to n'
         add leaf nodes of L which are flag on as
immediate antecedents of n'
         drop  leaf nodes of L which are marked
as flag off
         drop all the nodes subsumed by L until
n.conclusion
    call abstractProof on these new antecedents
of n'
match(n, r) :
// returns the nodes in n that are matched
   n: nodeset to be matched
   r: rule to be matched

   walk the tree for rule r to get a list L1 of
leaf nodes + conclusion
   walk the tree for nodeset n to get list L2 of
conclusion + antecedents

//try to match the conclusion of the rule with
all intermediate nodes of the proof
  for all elements i  of L2 except leaf nodes{
   if (unifiable(r.conclusion, i )) {
     int unifiedNodes = matchLeafNodes(i);
     if (L1.size == unifiedNodes);
          return L2
   }else{
     return [];
   }
matchLeafNodes(i):
 //matching continues for leaf nodes of L1
 //returns unified nodes count
walk the tree for nodeset i to get list L3 of
antecedents
  int unifiedNodes=0;
  for all elements q of L3{
    for all elements p of L1 {
   //try to match all leaf nodes of the rule with
      all nodes of the proof
     //this is for order independence
     if(unifiable(p, q))
       unifiedNodes++;
    }
  }
```

Frame 1. Abstractor Algorithm

Populating the AP Repository

The success of the approach depends on the size of the AP repository. Generic and domain-independent APs are ideal but domain-specific patterns can also be constructed and are sometimes quite useful for simplifying specific proofs. In this section we describe some APs already available in

the AP repository and their use in different contexts. The generic pattern shown here refers to the definition of transitivity. Several relationships possess such a property.

The class-subclass relationship, for instance, states that If X is a subclass of Y, and Y is a subclass of Z, then X is a subclass of Z. In a similar way the Instance-Class transitivity states that If X is a subclass of Y, and x is an instance of X, then x is also an instance of Y. Besides generic relationships, domain-specific ones are created as shown in the example below.

Case Study

We will describe how the abstraction approach has been used in the Knowledge Associates for Novel Intelligence (KANI) project within the DTO NIMD program. KANI supports the intelligence analysis task [12]. by helping analysts identify, structure, aggregate, analyze, and visualize task relevant information. It also helps them to construct explicit models of alternative hypotheses (scenarios, relationships, causality, etc.). As part of this effort, a Query Answering and Explanation component was developed that allows analysts to pose questions to the system. Answers are presented along with optional information about sources, assumptions, explanation summaries, and interactive justifications. In the KANI setting, not all data sources are reliable or current. Additionally, some information manipulation techniques (such as information extractors) may be unknown in advance. In this particular example, concepts such as person, office, owner and organization are involved in a reasoning process that aims at concluding the relationship between owners of an organization and their offices.

Figure 2 shows an example of abstraction where the domain-specific AP, named "Organization owner typically has office at the organization", can abstract away details of the proof. The Figure depicts the original piece of proof and the derivation done to generalize the proof. The ground axioms that the designer wished to maintain have been specified in the inference rule (in this case a direct assertion) that produces it. The abstracted template depicted in the upper part of Figure 2 describes the way an AP was generated from an original proof generated from JTP. Basically, there are two applications of modus ponens from direct assertions and implications that are simply saying that "an organization owner typically has an office at his/her organization". The abstracted proof is depicted in the right upper side. A possible explanation in English for the conclusion that "JosephGradgrind had an office at GradgrinFoods on April 1st, 2003" has been derived from the abstracted proof and it is presented in the right bottom.

Related Work and Discussion

Some proof transformation efforts share similar goals with our approach. Huang [8] studied resolution proofs in terms of meaningful operations employed by mathematicians. Huang argues that the transformation of machine-generated proofs in a well-structured natural deduction would help mathematicians comprehension. Our approach is designed to be used both by experts (e.g., logicians) and by lay-users. In this context Natural Deduction proofs are not abstract enough to provide users with an understanding of the reasoning process. Dahn and Wolf [3],[4] propose a variant of Natural Deduction called Block Calculus (BC) that can hide uninteresting formal sub proofs to facilitate the human users understanding.

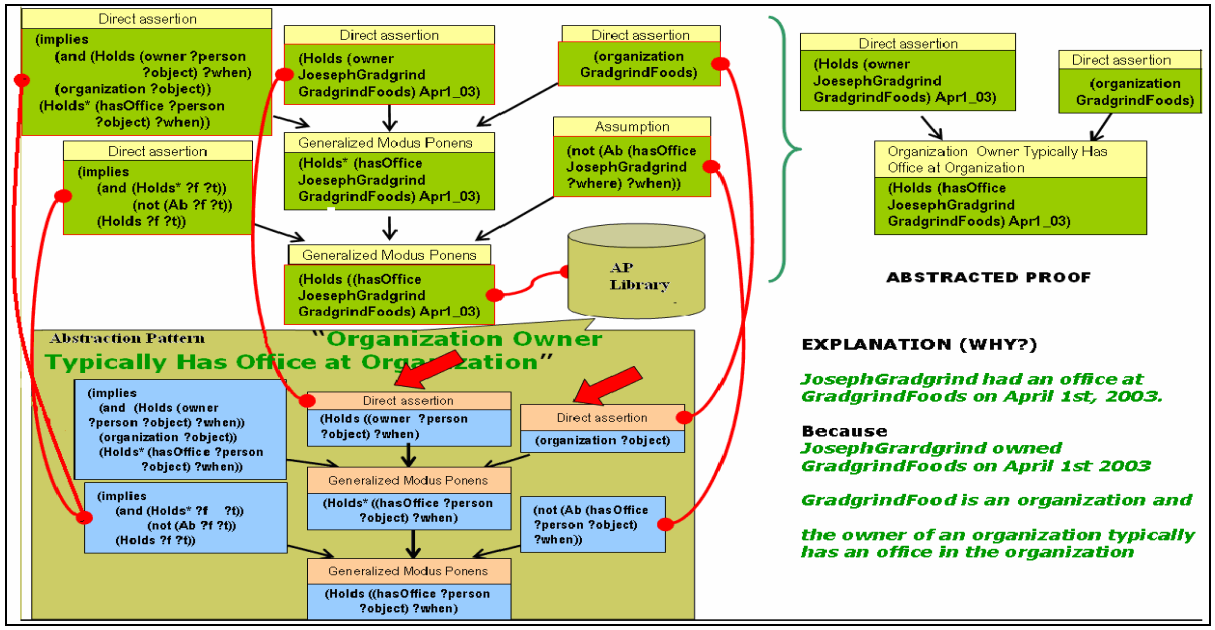

Figure 2 Example of utilization of domain-specific Abstracted Pattern.

The user can edit the simplified proof in the BC tool as a sequence of proof lines similar to the text in a text editor. However, this approach requires user edition for every proof and isn't appropriate in universal environments like Web. In our approach, the APs are designed by experts and the final users receive the abstracted proofs, according the APs that were used. Oliveira et al.[13] have shown that natural deduction proofs possess irrelevant information for explanations to human users. They define several strategies for proof simplification but their approach is restricted to these kinds of proofs and some strategies can only be applied in normalized proofs to guarantee that minimal formula fragments are obtained. Again the assumption of knowing about the features of the proof is impractical in the context of the complex distributed systems. Fiedler and Horacek [9] have considered that natural deduction proofs are very large with many irrelevant details that diminish human comprehension. They proposed an interactive method that provides increased or decreased detail depending upon the audience. Simple axioms assumed to be known by the audience are hidden. Such an approach has been used in the context of intelligent tutorial systems where it is possible to categorize the human user (i.e., the student). Such an assumption cannot be done in the context of the web. Denzinger and Schulz [5] presents a method for simplifying distributed proofs based on several heuristics as structural features (i.e. how isolated a sub proof is). We think that this approach is complementary to ours since we believe that pre-defined heuristics together with expert built patterns, like those we are proposing, are synergetic.

Our approach has a limitation of requiring a skilled designer for the abstraction patterns. The task of designing an AP is in some way subjective and the final explanation to a certain conclusion will strongly depend on the quality of these patterns.

Conclusion and Future Work

We described a generic approach for abstracting machine-level proofs generated from different reasoners on the web and encoded in PML into human-level justifications. Abstraction patterns used in the abstraction method are manually defined with the assistance of a set of tools to manipulate proofs in PML. From these patterns, the Abstractor algorithm walks along the proof tree matching the abstraction patterns and abstracting away irrelevant axioms thereby producing human-level explanations for answering queries.

We are studying ways to automatically prune axioms in the whole proof. Our idea is to propagate axiom elimination by the application of simplification strategies in certain axioms affected by specific inference rules. The or-inclusion in Natural Deduction Systems is an example of an inference rule that produces irrelevant axioms that can be eliminated from the entire proof. Typically, these axioms are inserted with the only goal of supporting resolution although they do not add any new information for explanation purposes.

References

[1] B. Buchanan, and E. Shortliffe. Rule based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project, Addison-Wesley, Reading, MA, 1984.

[2] W. Clancey. From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985, AI Magazine, 7(3), pp. 40-60, 1986.

[3] B. Dahn and A. Wolf. Natural Language presentation and Combination of Automatically Generated Proofs. Frontiers of Combining Systems,pp: 175-192, 1996.

[4] B. Dahn and A. Wolf. A Calculus Supporting Structured Proofs. Journal of Information Processing and Cybernetics, (5-6), pp: 262-276, 1994.

[5] J. Denzinger and S. Schulz. Recording and Analyzing Knowledge-based Distributed Deduction Process. Journal of Symbolic Computation, (11), 1996.

[6] C. Ghidini and F. Giunchiglia. A Semantics for Abstraction, European Conference on AI, 2004.

[7] X. Huang. Planning Argumentative Texts. In Proceedings of COLING94, Kyoto, 1994.

[8] X. Huang. Human Oriented Proof Presentation: A Reconstructive Approach. Ph. D Dissertation, DISKI 112, Saint Agustin, 1996.

[9] A. Fieldler, H. Horacek. Argumentation in Explanations to Logical Problems. Proceedings of ICCS, LNCS 2073, Springer Verlag, 2001.

[10] D. McGuinness and P. Pinheiro da Silva. Explaining Answers from the Semantic Web: The Inference Web Approach. Journal of Web Semantics.(1), n.4., pp: 397-413, 2004.

[11] D. McGuinness and P. Pinheiro da Silva. Infrastructure for Web Explanations. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), D. Fensel, K. Sycara and J. Mylopoulos (Eds.), LNCS 2870, Sanibel Is., FL, USA. Springer, pp: 113-129, 2003.

[12] J. Murdock, D. McGuinness, P. Pinheiro da Silva, C. Welty, and D. Ferrucci. Explaining Conclusions from Diverse Knowledge Sources. *Proc. of the 5th International Semantic Web Conference)*, pp:861-872, Athens, GA, 2006.

[13] D. Oliveira, C. de Souza, E. Haeusler. Structured Argument Generation in a Logic-Based KB-System. In Logic, Language and Computation, L. Moss, J. Ginzburg, M. de Rijke (eds). v. 2. CSLI Publication, 1999.

[14] M. Paterson and M. Wegman. Linear Unification. ACM Symposium of Theory of Computing, 181-186, 1976.

[15] V. Pinheiro, V. Furtado, P. Pinheiro da Silva, D. McGuinness. WebExplain: A UPML Extension to Support the Development of Explanations in the Web for Knowledge-Based Systems. Proc. Software Engineering and Knowledge Engineering Conference, San Francisco, 2006.

[16] W. Swartout, C. Paris, and J. Moore. Explanations in Knowledge Systems: Design for Explainable Expert Systems. IEEE Expert: Intelligent Systems and Their Applications, (6), n. 3, pp. 58-64, 1991.