User Interfaces for Portable Proofs

Paulo Pinheiro da Silva, Nicholas Del Rio

Department of Computer Science
University of Texas at El Paso
El Paso, TX, USA

Deborah L. McGuinness, Li Ding, Cynthia Chang

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, USA

Geoff Sutcliffe

Department of Computer Science
University of Miami
Coral Gables, FL, USA

Abstract

Portable proofs are a new and interesting way of integrating theorem provers into distributed environments like the web. This article reports on user interface's challenges and opportunities for theorem provers in such environments. In particular, this article reports on the design of user interfaces used for searching, browsing and inspecting TSTP problems when published as portable proofs.

Keywords: PML, TPTP, Inference Web, distributed proofs, user interfaces.

1 Introduction

The integration of theorem provers into hybrid distributed environments offers a new set of challenges and opportunities for providing explanations of system results. Distributed and portable proofs can be more interesting than stand alone proofs for a number of reasons: they may be deployed, stored and reused outside of environment in which they were generated; portions of the proof (e.g., individual inference steps or combinations of inference steps) may be named, annotated, and reused; support for portions of the proofs may be provided by other portions of the system (or even found by searching the web); axioms may have multiple lines of support; axioms can be asserted by multiple sources; and supporting evidence can be provided by multiple sources (instead of only the one source used in the original proof). We use the term *portable proofs* to refer to artifacts with these properties.

In order for portable proofs to realize their full potential, innovative user interfaces are required. For example, consider the following tasks:

- Searching for proofs and proof fragments
- Searching for proof annotations for reuse
- Browsing proof annotations

For example, what are the design requirements for the query interface of a proof-aware search engine? Also, what are the design requirements for the presentation of the search results? Considering that the results can be entire proofs or just proof fragments, how can a user interface show the exact part of the proof is represented by the search result? Moreover, how can the user intuitively ask for more details of the results, whether the additional results are related to in-depth disclosure of proof details or to a better understanding of the proof structure? The browsing of proof annotation can become particularly challenging considering the amount of details that can be incorporated into proofs.

In addition to the design issues above, we see that traditional challenges related to proof presentation remain for portable distributed proofs:

- Conclusion presentation
- Complex proof presentation
- Browsing techniques that incorporate evidence and sources

In this paper, we address the challenges above. The rest of this paper is organized as follows. Section 2 describes a typical use of our Inference Web explanation environment tool suite in a theorem prover setting. Section 3 revisits Inference Web's Proof Markup Language (PML) used to encode portable proofs. Section 4 explains how portable proofs are extracted from PML documents. Section 5 describes how Inference Web's Search (IWSearch) can be used to search for both proofs and proof metadata on the web. Section 6 introduces ProbeIt - a tool that supports proof inspection. Section 8 summarizes the main results for this paper.

2 Motivating Use Case

We leverage the TPTP collection of problems and proofs as the setting for our use case. Consider a simple scenario where a user is interested in solving one of the problems and investigating a particular theorem prover's solution. (Later we will expand to investigating multiple prover's solutions for the same problem). Our initial use case is the "Aunt Agatha" problem `PUZ001+1` in the TPTP collection [11], and consider the SNARK system's [10] solution of the problem.

Proofs generated by theorem provers can be published on the web. However, a typical proof output by a theorem prover is not annotated with meta information such as generator, time, and context. In fact, a proof's content is typically restricted to a raw identification of derivations plus a brief mention of the name of the inference rule used in each derivation. Without annotations, proofs may be used to debug the reasoning within theorem provers, but may be of limited use when trying to identify many other important properties of proofs such as the authors of the provers or a proper description of the inference rules used.

In this case of SNARK solving the Aunt Agatha problem, we may want to annotate that the proof was generated by SNARK. To be more specific, considering the possibility that the proof steps can be distributed on the web, we may want to annotate that SNARK was responsible for each step of the proof for Agatha. Further, we want the annotation to say that SNARK was implemented by Mark Stickel who is affiliated with SRI International. More generically, metadata should be able to be added to explain every single aspect of a proof, including the theorem provers responsible for generating the proofs, the version of the implementation, inference rules used by the theorem provers, axioms used in each proof, etc. More interestingly, metadata is expected to be reused at proof generation time. For example, an inference rule may be used multiple times in a proof as well as to be reused in multiple proofs from. In this case, one should be able to create and identifier, i.e., a URIref, and to publish the metadata about the rule. With this identifier in place, the metadata can be reused as needed.

We consider the following issues with relation to user interfaces for distributed proofs.

(i) How to search for proof-related metadata on the web, e.g., how to search for SNARK metadata?

(ii) How to verify that proof metadata correctly corresponds to the object of concern, e.g., that SNARK metadata is about the theorem prover from SRI International and implemented by Mark Stickel?

(iii) How to understand the structure of a distributed proof?

(iv) How to visualize a richly annotated proof?

The use of PML and (more) IW tools on the full TSTP solution library is also described in [8]. In the rest of the paper, we further describe the interface to the tools we use to create a demonstration environment for distributed proofs.

3 Proof Markup Language

In our environment, we encode distributed proofs in the Proof Markup Language (PML) [4,7]. We do this in the setting of the Inference Web [5] explanation infrastructure, which includes a number of PML-literate tools and services such as proof browsers, e.g., ProbeIt [1] and the IW Local View, and search services e.g., IWSearch. Inference Web also includes the PML ontologies and and references a collection of PML documents already available on the Web. We have generated a collection of PML proofs for the TPTP problems [8] and made the collection available on the Web.

Different than other markup languages for mathematical documents such as OMDoc [3], PML focus is on the creation and handling of graphs used to represent information manipulation traces created by agents (i.e., humans or machines) to infer conclusions. These graphs may be used to encode a formal proof but they may also be used to encode incomplete information on how conclusions were inferred. Moreover, a single graph may include a single justification for a given conclusion but it may include many alternate justifications for the same conclusion. Moreover, PML can be used to encode any kind of conclusion while OMDoc prescribes a precise way of encoding conclusions as formal logical sentences. Because of these characteristics, OMDoc is expected to have a better support for handling conclusions than PML since the conclusions need to conform to the OMDoc syntax. On the other hand, PML can be used to encode any kind of proof, including the proofs that can be encoded in OMDoc and informal proofs such as information extraction based on natural language processing [6].

In PML, NodeSet[1] and InferenceStep are the main constructs of portable proofs and web explanations.

A NodeSet represents a step in a proof whose conclusion is justified by any of a set of inference steps associated with the NodeSet. PML adopts the term "node set" since each instance of NodeSet can be viewed as a set of nodes gathered from one or more proof trees having the same conclusion.

- The `URIref`[2] of a node set is the unique identifier of the node set. Every node set has one well-formed URIref.
- The `hasConclusion` of a node set represents the expression concluded by the proof step. Every node set has one conclusion, and a conclusion of a node set is of type Information.
- The expression language of a node set is the value of the property `hasLanguage` of the node set in which the conclusion is represented. Every node set has one expression language, and that expression language is of type Language.
- Each inference step of a node set represents an application of an inference rule that justifies the node set's conclusion. A node set can have any number of inference steps, including none, and each inference step of a node set is of type InferenceStep. The inference steps are members of a collection that is the value of the property `isConsequentOf` of the node set. A node set without inference steps is of a special kind identifying an unproven goal in a reasoning process.

An InferenceStep represents a justification for the conclusion of a node set. Inference steps are anonymous OWL classes defined within node sets. For this reason, it is assumed that applications handling PML proofs are able to identify the node set of a inference step. Also for this reason, inference steps have no URIs.

- The rule of an inference step, which is the value of the property `hasRule` of the inference step, is the rule that was applied to produce the conclusion. Every inference step has one rule, and that rule is of type InferenceRule. Rules are in general specified by theorem prover developers. However, PML specifies three special instances of rules: *Assumption*, *DirectAssertion*, and *UnregisteredRule*. When specified in an inference step, the *Assumption* rule says that the conclusion in the node set is an explicit assumption. The *DirectAssertion* rule says that the conclusion of the node was provided by the sources associated with the inference step. The *UnregistredRule* says that the conclusion in the node set was derived by some unidentified rule. *UnregisteredRule*s allow the generation of proofs-like structures applying undocumented, unnamed rules.
- The antecedents of an inference step is a sequence of node sets each of whose con-

[1] PML concept names are typed in sans serif style and PML attribute names are typed in `courier` style.
[2] http://www.ietf.org/rfc/rfc2396.txt

clusions is a *premise* of the application of the inference step's rule. The sequence can contain any number of node sets including none. The sequence is the value of the property `hasAntecedent` of the inference step. The fact that the premises are ordered may be relevant for some rules such as *ordered resolution* [9] that use the order to match premises with the schema of the associated rule. For other rules such as modus ponens, the order of the premises is irrelevant. In this case, antecedents can be viewed as a set of premises.

- Each binding of an inference step is a mapping from a variable to a term specifying the substitutions performed on the premises before the application of the step's rule. For instance, substitutions may be required to unify terms in premises in order to perform resolution. An inference step can have any number of bindings including none, and each binding is of type VariableBinding. The bindings are members of a collection that is the value of the property `hasVariableMapping` of the inference step.
- Each discharged assumption of an inference step is an expression that is discharged as an assumption by application of the step's rule. An inference step can have any number of discharged assumptions including none, and each discharged assumption is of type Information. The discharged assumptions are members of a collection that is the value of the property `hasDischargeAssumption` of the inference step. This property supports the application of rules requiring the discharging of assumptions such as natural deduction's *implication introduction*. An assumption that is discharged at an inference step can be used as an assumption in the proof of an antecedent of the inference step without making the proof be conditional on that assumption.
- The engine of an inference step, which is the value of the property `hasInferenceEngine` of the inference step, represents the theorem prover that produced the inference step. Each inference step has one engine, which is of type InferenceEngine.
- The timestamp of an inference step, which is the value of property `hasTimeStamp` of the inference step, is the date when the inference step was produced. Time stamp is of the primitive type dateTime. Every inference step has one time stamp.

An inference step is said to be well-formed if:

(i) Its node set conclusion is an instance of the conclusion schema specified by its rule;

(ii) The expressions resulting from applying its bindings to its premise schemas are instances of its rule's premise schemas;

(iii) It has the same number of premises as its rule's premise schemas; and

(iv) If it is an application of the $DirectAssertion$ rule, than it has at least one source, else it has no sources.

PML node set schemas and PML inference step schemas are defined as follows. A PML **node set schema** is a PML node set which has a conclusion that is either a sentence schema[3] or a sentence; which has a set of variable bindings that map

[3] A sentence schema is a sentence optionally containing free variables. An instance of a sentence schema S

free variables in the conclusion to constants; which has zero of more inference steps; and whose inference steps are either inference steps or **inference step schemas**. An inference step schema is an inference set of a node set schema whose antecedents are node set schemas.

4 Portable Proofs

Since a PML node set can have multiple inference steps and each antecedent of each of those inference steps can have multiple inference steps, a PML node set N and the node sets recursively linked to N as antecedents of inference steps represent a graph of alternative proofs of N's conclusion. In this section, we describe how to extract individual proofs of N's conclusion from that graph of alternative proofs. We shall call each such extracted proof a "proof from N".

We begin by defining a *proof* as a sequence of "proof steps", where each proof step consists of a conclusion, a justification for that conclusion, and a set of assumptions discharged by the step. "A proof of C" is defined to be a proof whose last step has conclusion C. A proof of C is conditional on an assumption A if and only if there is a step in the proof that has A as its conclusion and "assumption" as its justification, and A is not discharged by a later step in the proof. An unconditional proof of C is a proof of C that is not conditional on any assumptions. (Note that assumptions can be made in an unconditional proof, but each such assumption must be discharged by a later step in the proof.) Finally, proof $P1$ is said to be a *subproof* of $P2$ if and only if the sequence of proof steps that is $P1$ is a subsequence of the proof steps that is $P2$.

Given these definitions, we can now define the proofs that are extractable from a PML node set as follows: for any PML node set N, P is a "proof from N" if and only if:

(i) The conclusion of the last step of P is the conclusion of N;

(ii) The justification of the last step of P is one of N's inference steps S; and

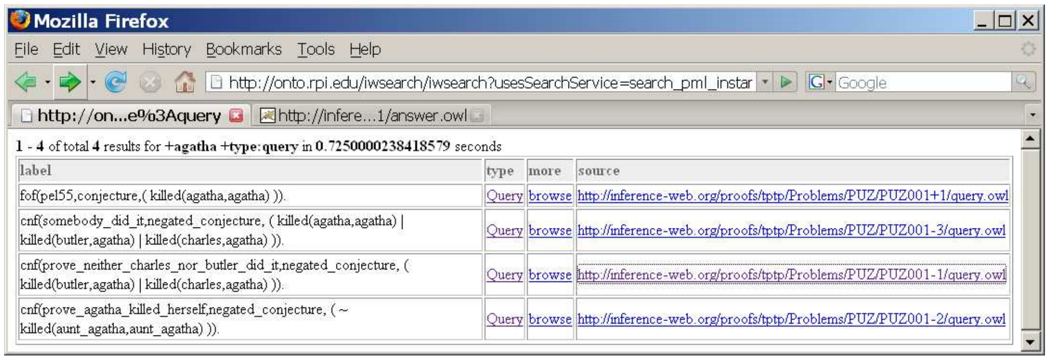
(iii) For each antecedent A_i of S, exactly one proof from A_i is a subproof of P.

If N is a node set with conclusion C, then a proof from N is a proof of C.

5 Searching for Proofs and Proof Metadata

IWSearch is the search tool for the Inference Web Infrastructure. IWSearch was developed to overcome a number of limitations related to metadata management found in our past practice: (i) IWBase, Inference Web's registry-based metadata management system, provides limited mechanisms for accessing metadata entries – a user can only browse the type hierarchy of those entries to find entries; and (ii) no service is available to find and reuse PML provenance metadata published on the web. IWSearch searches over PML proofs and proofs' metadata that has published on the web, and thus focuses on providing access to proof elements that have already been registered in the database registry.

is a sentence that is S with each free variable replaced by a constant.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://onto.rpi.edu/iwsearch/iwsearch?usesSearchService=search_pml_instar
Google
http://on...e%3Aquery
http://infere...1/answer.owl

1 - **4** of total **4** results for **+agatha +type:query** in **0.725000023841857 9** seconds

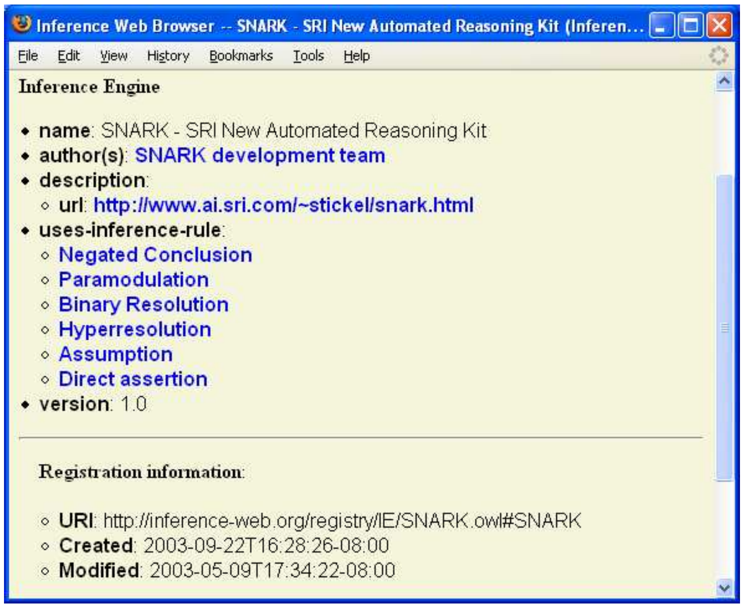
label	type	more	source
fof(pel55,conjecture,(killed(agatha,agatha))).	Query	browse	http://inference-web.org/proofs/tptp/Problems/PUZ/PUZ001+1/query.owl
cnf(somebody_did_it,negated_conjecture, (killed(agatha,agatha) | killed(butler,agatha) | killed(charles,agatha))).	Query	browse	http://inference-web.org/proofs/tptp/Problems/PUZ/PUZ001-3/query.owl
cnf(prove_neither_charles_nor_butler_did_it,negated_conjecture, (killed(butler,agatha) | killed(charles,agatha))).	Query	browse	http://inference-web.org/proofs/tptp/Problems/PUZ/PUZ001-1/query.owl
cnf(prove_agatha_killed_herself,negated_conjecture, (~ killed(aunt_agatha,aunt_agatha))).	Query	browse	http://inference-web.org/proofs/tptp/Problems/PUZ/PUZ001-2/query.owl

Fig. 1. IWSearch results for inference engine SNARK.

IWSearch is modeled off of SWOOGLE[2], which can be viewed as a search tool that “understands” RDF. Similarly, IWSearch can be viewed as a search tool that “understands” PML and OWL. IWSearch has an indexing phase that indexes terms, and also looks for particular terms using its knowledge of PML. Some metadata that IWSearch looks for includes:

- uri: Each PML object is identified by a unique identifier, i.e., a URI.
- type: Each PML object has one most-specific type, and IWSearch additionally indexes the other general types of a PML object. For example, an instance of inference engine metadata may also be considered as an instance of agent metadata.
- label: Each PML object has one label indicating its name. In the absence of name, the raw string content of the object is used. For example, an inference engine name is “SNARK 20070805r043”, but for a conclusion, its label is its raw string content -” ? [X] : (lives(X) & killed(X,agatha))”.
- source: Each PML object is extracted from one PML document, and the URL of the PML document is deemed as the source.

With the above metadata, IWSearch can provide much more than keyword search. By searching for `+SNARK +type:inferenceengine`, we can restrict the query and return only PML objects in the specified type. This is particularly useful if we want SNARK-generated proofs in PML to be annotated with the information that the proofs were generated by SNARK. Figure 1 shows the result of such a search. When querying for SNARK, one may find multiple metadata entries that are identified as SNARK. There are multiple reasons for this: more than one theorem prover is called SNARK; multiple versions of a single theorem prover; multiple metadata statements about the same theorem prover; or any combination of the previous reasons. By browsing the metadata, as in Figure 2, one may be able to verify multiple properties of the engine metadata such as authors, author’s affiliation, engine’s website as well as the creator of the metadata. By browsing the metadata, the user should be able to decide whether to reuse some existing metadata or even to create new metadata.


Inference Web Browser -- SNARK - SRI New Automated Reasoning Kit (Inferen...
File Edit View History Bookmarks Tools Help

Inference Engine

- **name**: SNARK - SRI New Automated Reasoning Kit
- **author(s)**: SNARK development team
- **description**:
  - url: http://www.ai.sri.com/~stickel/snark.html
- **uses-inference-rule**:
  - Negated Conclusion
  - Paramodulation
  - Binary Resolution
  - Hyperresolution
  - Assumption
  - Direct assertion
- **version**: 1.0

Registration information:

- **URI**: http://inference-web.org/registry/IE/SNARK.owl#SNARK
- **Created**: 2003-09-22T16:28:26-08:00
- **Modified**: 2003-05-09T17:34:22-08:00

Fig. 2. Browsing metadata about an inference engine called SNARK.

6 Browsing Proofs

Probe-It! consists of three primary views to accommodate the different kinds of proof information: queries, proofs (or justifications), and provenance (or metadata).

The `query view` shows the links between a given problem and possible solutions for the problem. Upon accessing one of the solutions in the query view, Probe-It! switches over to the `global view` associated with that particular solution. All views are accessible by a menu tab, allowing users to navigate back to the query view from any other view.

The `global view` graphically shows the reasoning associated with a given solution. Probe-It! renders this information either as a directed acyclic graph (DAG) or as a tree. The example of a tree view of the SNARK's solution for the Agatha problem is shown in Figure 3. In this view, users can visually see the conclusions of each node as well as some essential metadata.

The `local view` provides a comprehensive view of proof information available mainly at the level of a single proof step. For example, in Figure 4, one of the intermediate conclusions of the proof is that the butler hates himself ("hates(butler,butler"). The conclusion itself is encoded in `TPTP-CNF` language, and the view shows how the conclusion was derived: `SNARK 20070805r043` was the theorem prover responsible for deriving the conclusion by applying the rule `SNARK HYPERRESOLVE` to the antecedents also listed in the view. One of the main benefits of the global view is that it provides a good insight about the structure of the proof. For example, for the intermediate conclusion we can see that it was derived from three antecedents and that one of antecedents was itself derived from other statements. Further, the edge leaving the intermediate conclusion is evidence that it is not final (i.e., the intermediate conclusion is not an answer for the problem being solved by the inference engine.

The local view is structured to be a textual description of the main properties

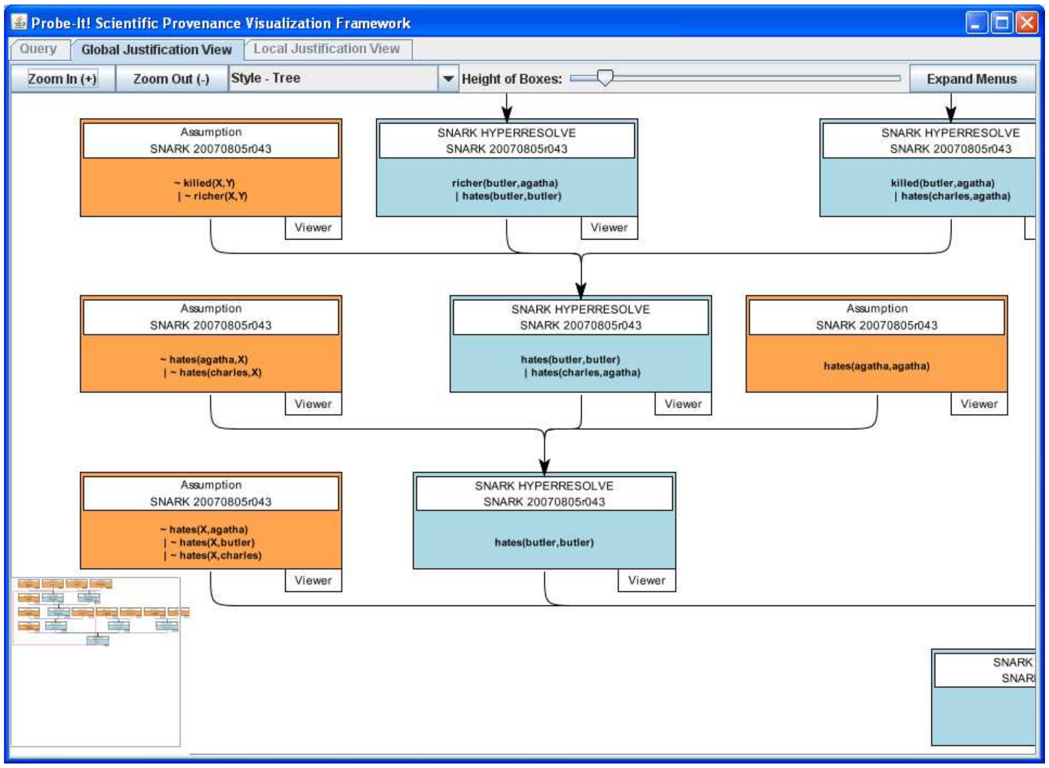
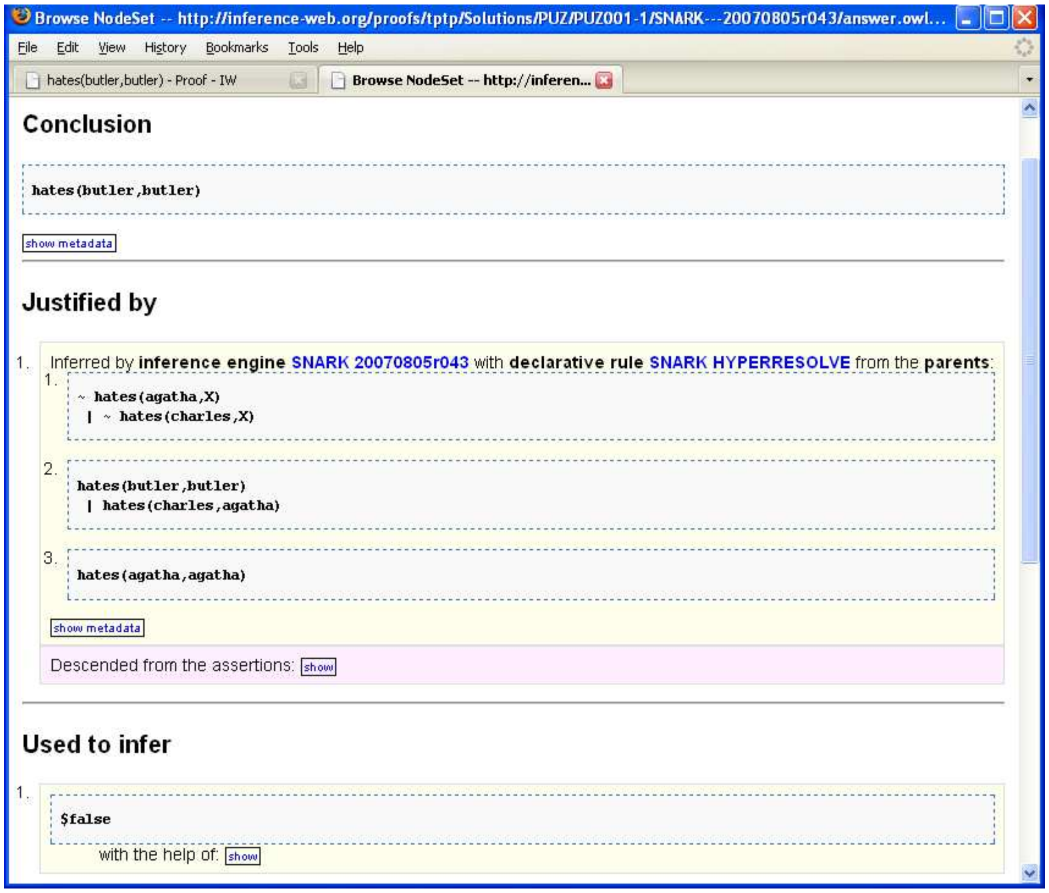


Fig. 3. ProbeIt! Global View

of a single proof step. This description is divided into four sections, as we can see in Figure 4: *conclusion*, *how*, *why*, and *to answer*. The conclusion section shows the main result of the selected inference step along with meta-information about the result. The how section identifies the antecedents as the inference rule applied to theses antecedents to infer the conclusion of the inference step. The why section shows the final conclusion of the entire proof and intermediate goals. The why section also identifies the following conclusion inferred from the conclusion of the current step of the proof. Last, the to answer section shows the question that the theorem prover is answering.

One very important aspect of the local view is that it provides information about sources and some usage information e.g., access time, during the execution of an application or workflow. Every node in the justification DAG has an associated provenance description. This information, usually textual, is accessible by selecting any of the aforementioned nodes. For example, upon selecting the "SNARK 20070805r043" hyper-link in the local view in Figure 4, meta-information about the inference engine, such as the responsible organization, is displayed in another panel. Similarly, users can access information transformation nodes, and view information about used algorithms. It is important to note that the requested meta-data is exactly the same information already presented in Figure 2. This exemplifies a case where user interface software can be reuse by multiple tools on the same way that the tools reuse meta-data to encode portable proofs.


Browse NodeSet -- http://inference-web.org/proofs/tptp/Solutions/PUZ/PUZ001-1/SNARK---20070805r043/answer.owl...
File Edit View History Bookmarks Tools Help
hates(butler,butler) - Proof - IW
Browse NodeSet -- http://inferen...

Conclusion

hates(butler,butler)

show metadata

Justified by

1. Inferred by **inference engine** SNARK 20070805r043 with **declarative rule** SNARK HYPERRESOLVE from the **parents**:

1. ~ hates(agatha,X) | ~ hates(charles,X)
2. hates(butler,butler) | hates(charles,agatha)
3. hates(agatha,agatha)

show metadata

Descended from the assertions: show

Used to infer

1. $false

with the help of: show

Fig. 4. ProbeIt! Local View

7 Implementation and Deployment

The core functionality provided by Probe-It! can be divided into three main subsystems: the PML API, the DIVA framework, and the visualization framework, which parse PML documents, provide a graphical framework from which execution traces can be rendered, and render the node set conclusions respectively. Both the PML API and the Diva framework are implemented in Java, while some viewers contained in the visualization framework require native libraries. XMDV, for example, is supported by OpenGL and both the 2D plot viewer and grid image viewer are based on native Generic Mapping Tools (GMT) scripts. Both the OpenGL and GMT libraries are implemented as Window's dynamic link libraries (DLLs). Although equivalent libraries for Linux and Macintosh exist, in the interest of time, only a Windows version was considered. The challenge of configuring Probe-It! to be compatible across all platforms will always exist because many of the popular viewers are pre-compiled commercial applications, that cannot be modified; instead, the current practice is to *wrap* these applications inside a Probe-It! by calling them from within Java.

Although Probe-It! contains a small set of pre-configured viewers, it is anticipated that Probe-It! will become more of a framework, from which scientists can subscribe existing viewers, thus difficulties with adapting Probe-It! to run on any

OS greatly restrict the portability of Probe-It!; we are in the process of implementing Probe-It! as a Web application.

8 Summary

Inference Web provides an explanation infrastructure for many types of distributed question answering systems, including theorem provers. It uses a proof interlingua called the Proof Markup Language as an explanation interchange language. It provides a collection of applications to handle proofs distributed on the web. Some of these applications are interactive tools that enable users to better visualize and thus understand portable proofs. In this paper, we provided a theorem prover style use case chosen from the TPTP library. We showed how the IWSearch tool may be used to find proofs with particular properties and provided an example from TPTP. We also described a use of ProbeIt for browsing portable proofs. ProbeIt allows theorem prover developers and users to visually inspect the structure of proofs (with the help of the global view) and the details of each node of a proofs (with the help of the local view).

The Inference Web infrastructure and framework is not restricted to a fixed number of tools to support a given functionality. For example, in terms of interactive tools in support of portable proof browsing, the Inference Web provides the following tools in addition to ProbeIt, as discussed in [8]: the original IWBrowser for browsing PML proofs and proof fragments with the help of standard HTML browsing capabilities and the NodeSet browser that has been integrated into ProbeIt in replacement to its original local view.

Acknowledgments

This work was supported in part by NSF grant HRD-0734825 and by DHS grant 2008-ST-062-000007.

References

[1] Del Rio, N. and P. Pinheiro da Silva, *Probe-it! visualization support for provenance*, in: *Proceedings of the Second International Symposium on Visual Computing (ISVC 2)* (2007), pp. 732–741.

[2] Ding, L., T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. C. Doshi and J. Sachs, *Swoogle: A search and metadata engine for the semantic web*, in: *Proceedings of the 13th CIKM*, 2004.

[3] Kohlhase, M., "An Open Markup Format for Mathematical Documents (Version 1.2)," Number 4180 in Lecture Notes in Artificial Intelligence, Springer Verlag, 2006.

[4] McGuinness, D., L. Ding, P. Pinheiro da Silva and C. Chang, *PML2: A Modular Explanation Interlingua*, in: *Proceedings of the AAAI 2007 Workshop on Explanation-aware Computing*, Vancouver, British Columbia, Canada, 2007.
URL http://www.ksl.stanford.edu/KSL_Abstracts/KSL-07-07.html

[5] McGuinness, D. L. and P. Pinheiro da Silva, *Explaining Answers from the Semantic Web*, Journal of Web Semantics **1** (2004), pp. 397–413.

[6] Murdock, J. W., D. L. McGuinness, P. Pinheiro da Silva, C. Welty and D. Ferrucci, *Explaining Conclusions from Diverse Knowledge Sources*, in: *Proceedings of the 5th International Semantic Web Conference (ISWC2006)* (2006), pp. 861–872.

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

[8] Pinheiro da Silva, P., 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*, Sydney, Australia, 2008, p. Accepted.

[9] Reynolds, J., *Unpublished seminar notes* (1966), stanford University, Stanford, CA.

[10] Stickel, M., *SNARK - SRI's New Automated Reasoning Kit*, http://www.ai.sri.com/ stickel/snark.html.

[11] Sutcliffe, G. and C. Suttner, *The TPTP Problem Library: CNF Release v1.2.1.*, Journal of Automated Reasoning **21** (1998), pp. 177–203.