PPDR: A Proof Protocol for Deductive Reasoning

Paulo Pinheiro da Silva[1] Patrick J. Hayes[2]
Deborah L. McGuinness[1] Richard Fikes[1]

[1]Knowledge Systems Laboratory, Stanford University
Stanford, CA 94305, USA.
e-mail: {pp,dlm,fikes}@ksl.stanford.edu
[2]Institute for Human and Machine Cognition
Pensacola, FL 32502
e-mail: phayes@ihmc.us

Abstract

The distributed and heterogeneous nature of the Web implies that a variety of agents may participate in helping to answer any question posed to a web application. Since multiple agents with various reasoning methods and representation languages may be used, inference rules used to derive any particular answer may be quite diverse. Since the evolving web is expected to continue to leverage diverse agents with diverse reasoning rules and since many users will need to have access to some explanation of how answers were obtained, this emerging web needs to have a way of leveraging and explaining hybrid reasoning. The emerging web needs an abstract, uniform way of specifying inference rules used in answer generation if tools are going to combine answers from hybrid systems, manipulate the justifications of those answers, and check to see if the justifications include correct applications of the inference rules. In this paper we introduce PPDR – the Proof Protocol for Deductive Reasoning – which is a language used to represent diverse inference rules. Also, in combination with the Proof Markup Language (PML), PPDR allows proofs produced by multiple agents to be combined, manipulated to present abstract explanations, and verified against the abstract rule specification.

1 Introduction

Distributed, hybrid applications such as those found on the Web benefit from the ability to have access to the inference rules[1] used to help determine answers

[1]In this paper, the term *inference rule* (or *rule*) refers to software implementations of primitive inferences used to derive conclusions from premises. Thus, inference rule refers to rules such as resolution and modus ponens rather than to theorems and assertions characterized,

or partial answers. If these applications have access to the inference rules or used to obtain answers, then they can enable features such as explanations of answers, combinations of partial answers from multiple agents, verification of inference rule application, etc. Beyond simple access to the inference rules, we would like to have a way of describing the inference rules in a declarative manner so that agents and humans can better understand the rules and can recognize patterns in the rules in order to facilitate abstraction and combination.

The work described in this paper presents an approach for specifying inference rules for the Semantic Web. The work is developed in the context of the Inference Web (IW) [9], which is an infrastructure for explaining answers from Semantic Web applications and services. Inference Web uses the Proof Markup Language (PML) [13], as its proof interlingua. PML can be used to represent the information agents need to understand results and the justifications of those results.

PML allows question answering systems to encode proofs of their answers. The proofs are described as collections of *NodeSet*s connected by *InferenceStep*s. A NodeSet represents a step in a proof whose conclusion is justified by all of the inference steps associated with the NodeSet. Note that there may be multiple ways to justify any one statement and each of those may be represented by an inference step associated with the NodeSet. An InferenceStep represents a justification for the conclusion of a node set. PML specifies the syntactic conditions under which a set of PML documents representing proofs is well-formed. One important requirement for having a well-formed set of PML documents is that inference steps must be correct applications of *inference rules* on inference step premises. This paper introduces PPDR – A Proof Protocol for Deductive Reasoning, which is a language for describing inference rules. This is intended to provide a way for rules to be declaratively represented and communicated, so that PML proofs can be checked for compliance against a set of rules.

PPDR is designed to be general-purpose in the sense that a variety of proof rule forms can be represented. For simplicity we require that the rules are stated with respect to a fixed logical language. We have chosen Simplified Common Logic (SCL)[2] as our logical notation. SCL is a modern successor to KIF [5] and provides a convenient, expressive language for our needs. The current standard semantic web languages and varieties of first-order logic can be translated into SCL.

The rest of the paper is organized as follows: Section 2 introduces the PPDR language specification. Section 3 describes an Inference Web use of PPDR for specifying inference rules as proof meta-information and of the use of these rule specifications in PPDR to check their correct applications in PML proofs. Section 4 describes related notation developments for declarative specifications of rules. Section 5 summarizes the main contributions of PPDR in the paper.

for instance, as production rules, business rules and event-condition-action rules.

[2]http://cl.tamu.edu/docs/scl/scl-latest.html

2 The Proof Protocol for Deductive Reasoning

2.1 SCL Schemas

An inference rule involves a general pattern of transformations on expressions, whereby parts of expressions are rearranged to produce a conclusion. PPDR uses *schemas* to state such transformations. We define a schema to be a pattern, which is any expression of SCL in which some lexical items of a certain grammatical category (typically things like `Sent`(ence), `Name`, `Rel`(ation symbol) etc.) have been replaced by a *schematic variable* (or meta-variable), paired with a set of syntax conditions, which record the corresponding type of each meta-variable (and possibly other conditions, described later). So, an SCL schema has the general form

pattern ;; *syntax-conditions*

where

- the *pattern* is an SCL expression with schematic variables. (PPDR also uses several other categories of meta-variable, noted below.) In order to distinguish schematic variables from SCL text in a pattern, we enclose literal SCL text in single quotes and leave schematic variables unquoted. This syntax is intended to indicate any piece of SCL core syntax text which can be obtained by substituting suitable lexical items for the schematic variables and concatenating the fragments in the order shown. Note that whitespace differences are ignored at the lexical syntax level. By convention, schematic variables must start with an alphabetic character and are written in lower case.

  Typically, rules involve applying substitutions to schematic variables. A *substitution* is a mapping from schematic variables to expressions of the appropriate type - more generally, which satisfies the syntactic conditions - and is represented by a list of pairs of variable, expression (e.g., v/t) (or by a schematic variable of the type substitution as discussed in Section 2.4.) A *substitution application* on an expression in a pattern is represented by a pair of square brackets following the expression and embracing the substitution. For example, $p[v/t]$ is a substitution meaning (whatever expression is substituted for) p with t substituted for v wherever it occurs in that expression.

- the *syntax-conditions* are specialized expressions specifying the types of the schematic variables. These are written in the SCL core syntax using a special vocabulary, but the meta-variables are treated here as normal variables.

For example,

'`(implies`' p q')';; (Sent p q)

is a schema which matches any SCL implication sentence. Here, p and q are schematic variables for SCL sentences. The syntax condition predicate `Sent` takes any number of arguments and is true just when they are all sentences.

2.2 List Notation

When stating rules it is often necessary to consider sequences of expressions, and SCL syntax is based on sequencing. We therefore allow another class of meta-variables which range over finite sequences, or lists, of expressions. For simplicity, we only allow sequences in which every expression has the same syntactic type, e.g. sequences of sentences or lists of terms. We use the simple but adequate convention that meta-variables starting with upper-case letters indicate expression sequences, while those starting with a lower-case letter indicate single expressions. Note that sequences have the structure of "flattened" lists, so that a single-element sequence can be identified with its single member, and concatenation is an associative operation which can be identified with LISP consing. PPDR uses use infix "+" as the concatenation operator and "$-$" as a binary sequence-difference operator. In addition, the infix dot notation acts as a selector, so that "$A.2$" refers to the 2nd item in the sequence A. The use of a variable name after the dot indicates an arbitrary element in the sequence. This use of variables over integers is the third kind of meta-variable used in PPDR pattern syntax. The notation "$(CardA)$" indicates the length of the sequence. The empty sequence has length zero and can be indicated by the notation "[ ]". With these conventions, for example, "$a + b$" indicates a sequence with two elements, "$a + C$" a sequence with at least one element and "$[a + b + C].2$" is b.

Using the list notation, we can specify new kinds of SCL schemas. For example,

'`(and`' N '`)`' ;; (Sent N)

is a schema which matches any SCL conjunction sentence.

The syntax conditions may use this sequence vocabulary, together with equality and other operations defined later, to state conditions which must be satisfied by the sentences involved in a rule. In general, syntax conditions can be viewed as computable conditions on bindings to meta-variables, and so any computable operation on SCL syntax or numerals is potentially usable in a PPDR syntax condition. While we anticipate that PPDR will be extended as needed by adding user-defined syntax conditions, the condition vocabulary described in this paper seems to be adequate to describe a wide variety of inference rules in existing logics.

2.3 Rule Schema

In PPDR compliant PML documents, a proof is a structure (not quite a tree) generated by rules which have the general form:

premise-list |$-$ *conclusion* ;; *syntax-conditions* where *premise-list* is a sequence of premises, separated by semi-colons, each of which is specified by a pattern and may be followed by an optional *discharged assumption*, written inside square brackets

> *premise*, [*discharged assumption*] ; ... ; *premise*, *discharged assumption*

and *conclusion* is also a pattern, and the syntax conditions apply to the whole rule. The following example

> **ndUI**: '`(forall` (' N ')' q ')' |– '`(forall` (' N - N.i ')' q[t/N.i] ')';;
> (Name N) (Sent q) (Term t)

is a rule schema for any SCL universal quantification. To check that the following rule is a correct application of `ndUI`:

```
(forall (a b c) d) |– (forall (a c) d['foo'/b])
```

it is sufficient to note a binding of expressions to the schematic variables which maps the schema to the rule while satisfying the syntactic conditions:

- N binds to $(a\ \ b\ \ c)$
- i binds to 2
- q binds to d
- t binds to foo

which satisfies the syntax conditions; and then the rule schema instantiates directly to:

```
(forall (a b c) d) |– (forall (a b c) - (a b c).2 d['foo'/
(a b c).2])
```

which in turn becomes the desired rule when the PPDR meta-notation is suitably evaluated following the equations $(abc).2 = b$ and $((abc) - (abc).2) = (ac)$:

```
(forall (a b c) d) |– (forall (a c) d['foo'/b])
```

2.4 Syntax Conditions

Grammatical Categories

Grammatical categories for variables in SCL schema patterns are specified by predicates which correspond to, or can be defined using, SCL grammatical categories.

- `Sent`(ence) is an SCL grammatical category;
- `Atom`(ic sentence) is an SCL grammatical category;
- `Lit`(eral) is an atom or a sentence of the form '`(not`' x ')' where (`Atom` x);
- `Name` is a variable when it occurs inside a quantifier that binds it;
- `Term` is an SCL grammatical category.

Unification Functions

The classical rule of *modus ponens* is easy to describe using the vocabulary defined so far:

> **ndMP**: p; '`(implies`' p q')' |– q ;; (Sent p q)

However, the more general rule *modus ponens with unification* (MPwU) requires us to introduce a new idea since its statement refers to a *unifier*:

> **MPwU**: p; '`(implies`' r q ')' |– q[s] ;; (Sent p q) (= s mgu(p,r))

In MPwU, s is a new schematic variable ranging over substitutions. A *substitution* is a mapping from variables to terms. The substitution notation $[v/t]$ already used denotes instances of such variables.

PPDR takes as primitive the most general simultaneous unifier function mgsu(). This takes as arguments two lists L and M of expressions and returns as value the most general substitution s such that $L.i[s] = M.i[s]$ for each i in the range $< 1, card(L) >$, if it exists. If M and L have different lengths then the result is undefined. The use of this function in a syntax condition asserts that the function is defined. The binary most general unifier of two expressions is the special case of mgsu() when its arguments are singleton sequences.

The following proof shows why the mgsu() is a useful primitive:

> **GMP**: A; '`(implies (and`' A ')' q ')' |– q ;; (Sent L q)

is an SCL proof schema for *generalized modus ponens* which allows an arbitrary number of premises. Thus, the following proof schema

> **GMPwU**: A; '`(implies (and`' B ')' q ')' |– q[s] ;; (Sent A B q)
> (= s mgsu(A,B)) (= card(A) card(B))

is an SCL proof schema for *generalized modus ponens with unification* that corresponds to MPwU for GMP. Here, s is defined over lists of sentences, A and B, rather than over a pair of sentences. Notice that if the antecedents are not unifiable then the schema does not apply.

Normal Form Sentences

Many inference engines normalize the sentences in proofs before applying their inference rules. PPDR schema can impose normal-form conditions by referring to grammatical categories and using schema description patterns directly in the description of the syntax conditions. For example, the following proof

> **BiRes**: '`(or`' A ')', '`(or`' B ')' |– '`(or`' A + B - A.i - B.j ')' ;;
> (Lit A B) (= A.i '`(not`' B.j')')

is an SCL proof schema for bi-resolution applied to clauses represented in SCL as disjunctions of literals.

2.5 Sentence Discharge

The following proof

ndImplIntro: p, [q] |– '(implies ' q p ')' ;; (Sent p q)

shows that *q* was discharged in order to introduce the implication in the proof schema conclusion. Moreover, the proof shows that *q* was an assumption for *q*. For natural deduction *or-elimination* we have the following:

ndOrElim: '(or' p q ')' ; r, [p]; r, [q] |– r ;; (Sent p q r)

where *p* is an assumption for the first *r* premise while the *q* is an assumption for the other *r* premise.

2.6 PPDR Specifications in XML

The notation presented so far is the human-friendly version of PPDR. In fact, the rules specifications presented in this paper show that even complex rules such as GMPwU and ndOrElim can be shortly represented in PPDR. Considering the Semantic Web use of PPDR, rule specifications can also be represented in XML. For example, the PPDR specification for ndUI introduced in Section 2.3 can be represented in XML as follows:

```
<ppdr:Rule name="ndUI">
     <ppdr:Premise syntax="scl">
        (forall ( <ppdr:var type="nameList">N</ppdr:var> )
         <ppdr:var type="sentence">q</ppdr:var>
        )
     </ppdr:Premise>
     <ppdr:Conclusion syntax="scl">
         (forall (
         <ppdr:op type="removeItemFromList">
              <ppdr:var>N</ppdr:var>
              <ppdr:op type="selectItemInList">
                 <ppdr:var>N</ppdr:var>
                <ppdr:var type="index">i</ppdr:var>
             </ppdr:op>
          </ppdr:op>
               )
           <ppdr:op type="instantiate"
              <ppdr:var> q </ppdr:var>
              <ppdr:substitution>
                <ppdr:var type="term">t</ppdr:var>
                <ppdr:op type="selectItemInList">
                  <ppdr:var>N</ppdr:var>
                  <ppdr:var>i</ppdr:var>
                </ppdr:op>
              </ppdr:substitution>
            </ppdr:op>
        )
     </ppdr:Conclusion>
</ppdr:Rule>
```

Premises and conclusions are patterns written in the syntax form indicated. Then all patterns are SCL syntax written as body text, with marked-up <ppdr:var> and <ppdr:op> items in it, each with a property indicating type. Properties are the grammatical categories of PPDR as described in Section 2.4 with a distinction, at the XML level, between single elements (i.e., Term) and lists (i.e., TermList). For example, the statement <ppdr:var type=''SentenceList''>N</ppdr:var> says that N is a SentenceList. Meta-operations are expected to be applied during matching of the pattern to a rule. Meta-operators such as $N - N.2$ are handled by special markup, consisting of operators applied to arguments as in the following example:

```
<ppdr:op type="removeItemFromList">
    <ppdr:var>N</ppdr:var>
    <ppdr:op type="selectItemInList">
       <ppdr:var>N</ppdr:var>
       2
    </ppdr:op>
</ppdr:op>
```

3 Inference Web and PPDR

Inference Web supports the use of any language for writing proof node conclusions and for specifying inference step rules. Moreover, it supports the registration of multiple languages for representing expressions and expression specifications. Section 3.1 describes how PPDR expressions can be used to specify inference rules registered in a repository of proof-related meta-data. Section 3.2 describes how rule specifications written in PPDR and stored in the meta-data repository can be used for checking the correctness of rule applications on PML proofs. In order to use PPDR in proofs where expressions are written in languages other than SCL, Section 3.3 describes the Inference Web support for translation rules.

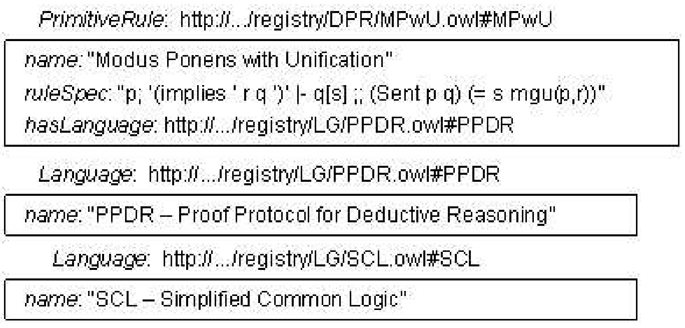
3.1 PPDR Specification of Rules in IWBase

When presenting a typical proof, an engine may state which inference rule was applied to some premises to infer the proof conclusion. Premises and conclusions are main elements of proofs which are connected by inference rules applied to premises producing conclusions. Typical proofs, however, rarely include rule specification information.

Meta-information can be used to enhance proofs. In the Inference Web, IWBase [10] is a distributed repository of meta-information providing services for maintaining entries and for coordinating the distributed nodes of the repository. Figure 1 shows the registration of the MPwU rule[3] in IWBase having the following attributes:

- a URI that is the unique identifier for the rule – `http://.../registry/DPR/MPwU.owl #MPwU`;
- a type – *PrimitiveRule*;
- a name – "Modus Ponens with Unification";
- a string containing the rule's formal specification; and
- a representation language used for writing the rule specification.

Boxes in the figure are abstractions of PML documents written in OWL and representing meta-level concepts related to proofs. Each PML concept has a type and a URI identified above each box.



PrimitiveRule: http://.../registry/DPR/MPwU.owl#MPwU

name: "Modus Ponens with Unification"
ruleSpec: "p; '(implies ' r q ')' |- q[s] ;; (Sent p q) (= s mgu(p,r))"
hasLanguage: http://.../registry/LG/PPDR.owl#PPDR

Language: http://.../registry/LG/PPDR.owl#PPDR

name: "PPDR – Proof Protocol for Deductive Reasoning"

Language: http://.../registry/LG/SCL.owl#SCL

name: "SCL – Simplified Common Logic"

Figure 1: IWBase proof meta-information.

A few points worth noting follow:

- meta-information related to objects referred to in proofs such as rules are registered in the IWBase. That meta-information can be used anytime for several purposes including checking rule applications as discussed in Section 3.2. Figure 1 also shows a few representation language entries used for specifying proof-level contents, i.e., the entry for the SCL and PPDR languages.

[3]The actual IWBase entry for MPwU may be visualized online using an IWBase registrar at http://iw.stanford.edu/iwregistrar or it can be accessed directly from the IWBase registry in its original OWL [11] format at http://iw.stanford.edu/registry/DPR/MPwU.owl.

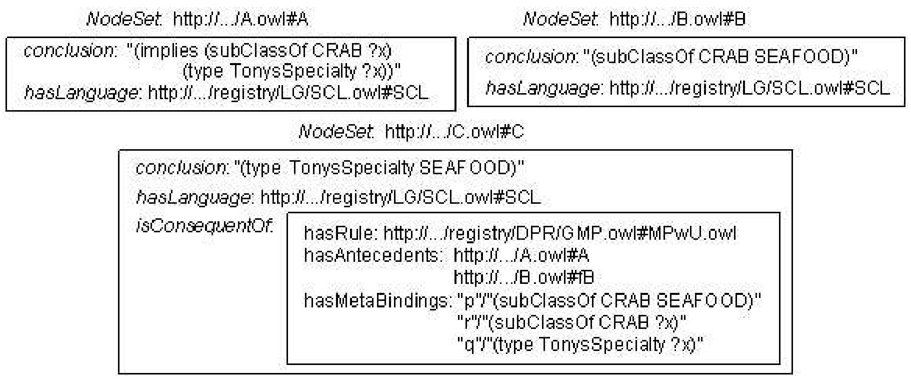

NodeSet: http://.../A.owl#A
conclusion: "(implies (subClassOf CRAB ?x) (type TonysSpecialty ?x))"
hasLanguage: http://.../registry/LG/SCL.owl#SCL
NodeSet: http://.../B.owl#B
conclusion: "(subClassOf CRAB SEAFOOD)"
hasLanguage: http://.../registry/LG/SCL.owl#SCL
NodeSet: http://.../C.owl#C
conclusion: "(type TonysSpecialty SEAFOOD)"
hasLanguage: http://.../registry/LG/SCL.owl#SCL
isConsequentOf:
hasRule: http://.../registry/DPR/GMP.owl#MPwU.owl
hasAntecedents: http://.../A.owl#A
http://.../B.owl#B
hasMetaBindings: "p"/"(subClassOf CRAB SEAFOOD)"
"r"/"(subClassOf CRAB ?x)"
"q"/"(type TonysSpecialty ?x)"

Figure 2: PML proof-level information.

- PPDR may be used to specify inference rules at the proof meta-level. Thus, tools can use these specifications for several reasons including proof transformations based on the matching of derived rules against proof fragments. Since proof transformation may be used to abstract proofs into more meaningful explanations, this can be of value.

3.2 PML-Based Inference Step Checking

PPDR may be used to verify whether inference steps in PML documents are correct applications of inference rules. Figure 2 shows a PML proof fragment composed of three node sets, A, B and C, where we want to check if the inference step in node C is a correct rule application. So, each node set is represented as a stand-alone rectangle[4] that is uniquely identified by a URI. The URI for A is `http://.../A.owl#A` and the URIs for B and C are similar to the URI for A replacing "`A`" by "`B`" and "`A`" by "`C`" respectively. The inference step to be checked is represented as a box within C attached to the `isConsequentOf` property. The inference step is said to be an application of MPwU as indicated by the URIref in the `hasRule` property of the inference step. Thus, the step is a correct application of MPwU if:

1. the conclusions of the inference step antecedents, which are `(implies (subClassOf CRAB ?x) (type TonysSpecialty ?x))` and `(subClassOf CRAB SEAFOOD)`, match the rule premises, which is "`p; '(implies ' r q ')'`";
2. the inference step conclusion, which is `(type TonySpecialty SEAFOOD)`, matches the rule conclusion, which is "`q[s]`";

[4]The graphical notation in Figure 2 is consistent with the notation in Figure 1 since boxes on both figures are PML *elements* as described in [13]. Boxes in the Figure are abstractions of PML documents written in OWL. Stand-alone boxes represent PML node sets and the inner-box represents a PML inference step. The URI of each node set is identified on its top.

3. the rule syntactic conditions can be verified for conditions 1 and 2.

From the `hasMetaBindings` property of the inference step, we know that: p is bound to `(subClassOf CRAB SEAFOOD)`; r is bound to `(subClassOf CRAB SEAFOOD)`; and q is bound to `(type TonysSpecialty ?x)`. Thus, the inference step meets requirement 1 since it has two premises and the premise that is not bound to p is an implication (it has the SCL lexicon "implies"). We can also verify that the step meets the requirement 2 since the conclusion and the sentence derived from the implied part of the premise has the same predicate (i.e. "type") and both have two arguments. Thus, $mgu(p/r) =$"SEAFOOD"/?x. Moreover, the inference step is a correct application for MPwU since the conclusion of C is the application of $q[mgu(r, p)]$.

The `hasMetaBindings` property of inference step assumes that inference engines producing PML documents have access to the rule specification in the IWBase in order to generate the meta-level bindings. PML, however, does not force inference steps to have meta-level bindings in order to be well-formed. Therefore, checking tools may be unable to decide the correct way of binding schematic variables if meta-level bindings are not provided.

In this paper we use the term *inference step checking* rather then *proof checking* since a comprehensive checking of proofs in PML documents may involve other tasks and is beyond the current scope of our work. For instance, it may involve a syntactic verification of the PML documents. Moreover, for the conclusion of each node, it may involve the verification if they are valid expressions according to the language specified by the `hasLanguage` property of the node set.

3.3 IWBase Translation Rules

In addition to primitive and derived rules, IWBase supports the registration of translation rules representing expression transformations from one language into another one. Translation rules are directional so a rule from A to B and a rule from B to A are different. IWBase restricts to one the number of translation rules from one given language into another given language.

A translation rule associated program be called during the process of checking a proof if registered in IWBase. Registered associated programs (translators) can either translate an expression written in A into an expression written in B or return an error code informing that it cannot perform the translation. In fact, translators are not required to support full translation between languages.

PML checking tools based on PPDR can always try to translate sentences written in languages other than SCL into SCL by used translation rules and their translators. Thus, once node set conclusion can be translated into SCL, checking tools can verify if rule specifications in PPDR can match inference step premises and conclusions as described in Section 2.3. SCL is used as a base language for PPDR since SCL it is an abstract language and translators from first-order languages into SCL are usually easy to implement.

4 Related Work

Inference rules are often implemented rather than specified declaratively. Some systems, however, allow users to specify inference rules. For instance, Isabelle [12], as one of the many theorem provers that adopted Edinburgh LCF [6] techniques of programming inference rules, is an inference engine where users can specify their own rules (LCF and Isabelle are also called programmable theorem provers). Although Isabelle is a powerful engine in the sense that it supports a wide range of kinds of inference rules and logics, its rule specifications are neither abstract nor declarative. Thus, using the LCF meta-language, users need to program, for example, how a specific unification should be implemented. Moreover, it may be very difficult and sometimes impossible to combine Isabelle proofs depending on which set of rules was used to generate each proof. Nevertheless, rule specifications in Isabelle are not intended to be used for matching logical sentences written in representation languages other than the one specified in Isabelle.

JAPE [14] is a proof editor where users can interact with the editor to create or modify proofs. Proofs are created and modified according to abstract rule specifications [4]. The JAPE notation for specifying rules has many similarities with PPDR including syntax conditions that are called provisos. One difficulty in using the JAPE meta-language is that it requires encoding syntactic categories for every new language used for representing logical sentences. So, it does have limited capability for identifying different classes of expressions however, it may not be considered simple to use this capability.

Similar to PPDR, the OWL Rule Language (ORL)[7] is another rule specification language also intended to be used for the Semantic Web. ORL claims to add expressive power to OWL and to be a syntactical and semantical extension of OWL itself. Also, ORL rules are basically represented in OWL. The "human readable" version of ORL as described in the paper is too simple to accommodate the complexity of rules as described in this paper. Moreover, ORL does not address the problem of integrating proofs provided from distinct engines and with expressions represented in different languages.

The current PPDR is expected to be extended to incorporate some of the useful aspects of the rule specification languages described above. For example, we expect PPDR to become as abstract as the JAPE notation but without having the same difficulties of encoding syntactic categories for several representation languages. The IWBase may be a possible alternative to avoid the registration of new syntactic categories if a wide number of rules can be translated into a common language such as SCL. As opposed to ORL, PPDR does not aim to have a unified representation for "proper" inference rules (the primitive rules implemented in inference engines) and rules derived from primitive rules. In fact, the OWL rule language assumes that PML can be used to describe derived rules and theorems can be described as derived rules. Therefore, rules called "production rules", "business rules" and "event-condition-action rules" can be defined in terms of PML proofs even without the need of PPDR.

There is also related work in proof transformation, proof rewriting, and matching. The closest to our work is [1] on matching patterns initially used for pruning explanations of earlier description logics [8, 3] and then later rewriting work [2]. This work was aimed initially at matching for pruning, then matching more generally, and in term rewriting for non-standard inferences and finding unifiers. Moreover, the initial focus on the earlier work was to represent patterns to match concept descriptions (for presentation and pruning) and the focus on this work is to represent patterns in rules (for both abstraction and rule combination).

5 Conclusions

In this paper, we have introduced PPDR - a Proof Protocol for Deductive Reasoning. PPDR provides a language that supports distributed hybrid question answering. By providing a language for encoding inference rules, it facilitates proof combinations (thereby supporting interoperability) inference rule specification (thereby supporting justification access and presentation), pattern specification (thereby supporting abstraction and matching), and proof checking (thereby improving checking and reliability of question answering systems).

PPDR provides a language that supports abstract, uniform encodings of inference rules. Particularly in combination with the Proof Markup Language, it provides a flexible and necessary foundation enabling proof presentation, abstraction, and combination thereby providing an infrastructure for interactive and interoperable explanations of answers from hybrid systems.

References

[1] Franz Baader, Ralf Küsters, Alexander Borgida, and Deborah L. McGuinness. Matching in Description Logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.

[2] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting Concepts Using Terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[3] Alex Borgida and Deborah L. McGuinness. Asking Queries about Frames. In *Proceedings of Fifth International Conference on the Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, November 1996. Morgan Kaufmann.

[4] Richard Bornat and Bernard Sufrin. *Roll your own JAPE logic*, jape version 3.2 edition, September 1997.

[5] Michael R. Genesereth and Richard Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA, USA, 1992.

[6] Michael J. C. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation.* Number 78 in LNCS. Springer-Verlag, 1979.

[7] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004).* ACM, 2004. To appear.

[8] Deborah L. McGuinness. *Explaining Reasoning in Description Logics.* PhD thesis, Rutgers University, 1996.

[9] Deborah L. McGuinness and Paulo Pinheiro da Silva. Infrastructure for Web Explanations. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, LNCS-2870, pages 113–129, Sanibel, FL, USA, October 2003. Springer.

[10] Deborah L. McGuinness and Paulo Pinheiro da Silva. Registry-Based Support for Information Integration. In *Proceedings of IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pages 117–122, Acapulco, Mexico, August 2003.

[11] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), December 9 2003. Proposed Recommendation.

[12] Lawrence C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[13] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard Fikes. A Proof Markup Language for Semantic Web Services. Technical Report KSL-04-01, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, January 2004.

[14] Bernard Sufrin and Richard Bornat. Encoding a natural deduction system for the jape proof editor. Technical Report PRG-TR-9-98, Programming Research Group, Oxford University Computing Laboratory, 1998.