Towards Checking Hybrid Proofs

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

[1]Knowledge Systems Laboratory, Stanford University
Stanford, CA 94305, USA.
e-mail: {pp,dlm,fikes,deshwal}@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 today's applications such as the Web implies that a variety of agents may participate in answering questions. Since multiple agents with various reasoning methods and representation languages are possible, inference rules used to derive any particular answer may be quite diverse. In this paper we introduce the Inference Meta Language to represent inference rules in an abstract and uniform way. The language may be used to annotate proofs enabling them to be automatically checked. Checking may be critical if systems combine answers, i.e., web service composition. We have implemented a parser and checker for the language and it is in use in several proof-based explanation solutions.

1 Introduction

Distributed, hybrid applications such as those found on the web often use information from other agents and services to produce their own answers. These applications may need input information to be accompanied by proof information if they are going to accept and trust a result produced by an unfamiliar service. For example, if a user or application has a reason to suspect that an answer may not be reliable, we would like to enable the user or application to inspect information about how the answer was obtained. Further, the application may want to actually check the proof to determine at least that the agents involved with the proof generation are not working incorrectly. Obtaining information that a proof is not incorrect will not necessarily guarantee a correct answer, *but* an incorrect proof will definitely increase an application's suspicion about the answer. In order to check proofs, proof checkers need to understand and use the proof format; proofs need to identify the inference rule applied in each proof step; and proof checkers need to have information that enables them to check the identified inference rules. By using a common proof format such as the Proof Markup Language (PML) [17] or PDS [5] proof checkers can understand and use proofs. In order for proof checkers to check an inference rule application, they either need to be familiar with the rule or they need to have access to a specification of the rule they are supposed to check.

As a simple example, consider an application that will depend on answers from the Wine Agent [10]. The application may know very little about the Wine Agent and thus it would like to know if information that the wine agent derived is something that it should suspect. For example, the Wine Agent might claim that TonysSpeciality is a seafood dish and it is appropriately paired with a particular white wine. The application might want to know how the application determined that TonysSpeciality was a seafood dish. This could have been simply told to the wine agent (in which case the application might want to know information about what source stated this (e.g., [18]) and if that source is viewed as trusted (e.g., [20])), it might have been the result of an extraction engine (in which case the application might want to know what raw sources were used, what extraction inferences were used (e.g., [15]) and if they were applied correctly, etc.) and it might have been the result of applications of standard first order logic reasoning inferences, for example, applied by the JTP hybrid reasoning system [6]. In the last case, the application might also want to know what inferences were applied, which reasoners applied them, if those inferences were applied correctly.

In this paper, we address the issue of helping applications evaluate if inference rules were applied correctly in individ-

ual applications and in hybrid applications - those that use more than one agent to obtain an answer. A proof checker that already has an understanding of the inference rules that it will be asked to verify could do a portion of this task. Our work also enables proof checkers to access specifications of new inference rules so that they may also check for correctness of these new rules. This specification of inference rules also enables additional features such as explanations of answers and combinations of partial answers from multiple agents, 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 checking hybrid proofs. The work is developed in the context of the Inference Web (IW) [12], which is an infrastructure for explaining answers from web applications and services. Inference Web uses PML, as its interlingua for distributed, hybrid proofs. PML can be used to represent the information agents need to understand results and the justifications of those results.

The rest of the paper is organized as follows: Section 2 describes the Proof Markup Language used in this paper to encode hybrid proofs. Section 3 introduces the InferenceML language specification used to annotate hybrid proofs with inference rule specifications. Section 4 describes an infrastructure for specifying, maintaining and using inference rules as proof meta-information. Section 5 describes the use of InferenceML rule specifications to check their correct applications in PML proofs. Section 6 describes related notation developments for declarative specifications of rules. Section 7 summarizes the main contributions of InferenceML in the paper.

2 Hybrid Proofs

PML allows question answering systems to encode proofs of their answers and to use answers from other question answering systems by combining proofs. A proof is said to be hybrid if parts of the proofs are produced by distinct agents. In terms of PML concepts, 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. The next section introduces the Inference Meta Language (InferenceML) for describing inference rules since PML itself does not require one particular language. Thus, an inference step may point to a rule entry in a metadata repository, as describe in Section 4.1, where tools can access the rule specification and the language used to specify the rule. 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.

3 The Meta Language for Inference Rules

InferenceML 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)[1] as our logical notation. SCL is a modern successor to KIF [7] 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.

3.1 SCL Schemas

An inference rule involves a general pattern of transformations on expressions, whereby parts of expressions are rearranged, potentially normalized and expanded, to produce a conclusion. InferenceML 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 syntactical 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. (InferenceML 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 that can be obtained by substituting suitable lexical items for

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

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 3.4.) A *substitution* applied to 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 applied to p with t substituted for v wherever v occurs in

- 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 as normal variables.

For example,

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

is a schema that 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 exactly when the arguments are sentences.

3.2 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 that 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 that can be identified with LISP consing. InferenceML 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 the InferenceML pattern syntax. The notation "$(CardA)$" indicates the cardinality or number of elements in the sequence. The empty sequence has cardinality 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 that matches any conjunctive sentence in SCL.

Conditions may be specified using this sequence vocabulary, together with equality and other operations defined later, to state conditions that 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 an InferenceML syntax condition. While we anticipate that InferenceML 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.

3.3 Rule Schema

In InferenceML compliant PML documents, a proof is a structure (not quite a tree) generated by rules that 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

ndUQ: '(forall ('N')'q')'$|-$'(forall (' N - N.i ')' q[t/N.i] ')';; (Name N) (Sent q) (Term t)

is a rule schema for any SCL natural deduction universal quantification (ndUQ). To check that the following rule is a correct application of `ndUQ`:

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

it is sufficient to note a binding of expressions to the schematic variables that 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 InferenceML 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])
```

3.4 Syntax Conditions

Grammatical Categories

Grammatical categories for variables in SCL schema patterns are specified by predicates that 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.

InferenceML 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* that 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. InferenceML 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.

3.5 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.

3.6 InferenceML Specifications in XML

The notation presented so far is the human-friendly version of InferenceML. In fact, the rules specifications presented in this paper show that even complex rules such as GMPwU and ndOrElim can be shortly represented in InferenceML. Considering the Semantic Web use of InferenceML, for example, rule specifications can also be represented in XML. For example, the InferenceML specification for ndUQ introduced in Section 3.3 can be represented in XML as follows:

```
<infml:Rule name="ndUQ">
 <infml:Premise syntax="scl">
    (forall (
     <infml:var type="nameList">
        N
     </infml:var> )
     <infml:var type="sentence">
        q
     </infml:var>
    )
 </infml:Premise>
 <infml:Conclusion syntax="scl">
   (forall (
    <infml:op type="removeItemFromList">
       <infml:var>N</infml:var>
       <infml:op type="selectItemInList">
          <infml:var>N</infml:var>
          <infml:var type="index">
             i
          </infml:var>
       </infml:op>
     </infml:op>
        )
     <infml:op type="instantiate"
        <infml:var> q </infml:var>
        <infml:substitution>
          <infml:var type="term">
             t
          </infml:var>
          <infml:op type="selectItemInList">
            <infml:var>N</infml:var>
            <infml:var>i</infml:var>
            </infml:op>
       </infml:substitution>
     </infml:op>
     )
 </infml:Conclusion>
</infml: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 <infml:var> and <infml:op> items in it, each with a property indicating type. Properties are the grammatical categories of InferenceML as described in Section 3.4 with a distinction, at the XML level, between single elements (i.e., Term) and lists (i.e., TermList). For example, the statement <infml:var type=''SentenceList''>N</infml:var> says that N is a SentenceList. Meta-operators 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:

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

4 Infrastructure for Checking Hybrid Proofs

Inference Web supports the registration of multiple languages for representing expressions and expression specifications. Section 4.1 describes how InferenceML expressions can be used to specify inference rules registered in a repository of proof-related meta-data. In order to use InferenceML in proofs where expressions are written in languages other than SCL, Section 4.2 describes the Inference Web support for translation rules.

4.1 InferenceML 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 that 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 [13] 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[2] 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.

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. Figure 1 also shows a few representation language entries used for specifying proof-level contents, i.e., the entry for the SCL and InferenceML languages.
- InferenceML 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.

4.2 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.

[2]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 [14] format at http://iw.stanford.edu/registry/DPR/MPwU.owl.

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 InferenceML 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 InferenceML can match inference step premises and conclusions as described in Section 3.3. SCL is used as a base language for InferenceML since SCL it is an abstract language and translators from first-order languages into SCL are usually easy to implement.

5 Checking Hybrid Proofs

A generic checking service for PML proofs first checks all the inference steps in proofs and second, for all proof ground assertions, it makes use of the source usage feature of PML to verify that assertions actually match source contents. The checking service works in the same way whether the proof is hybrid or not and follows the steps described in Section 5.1. In fact, the checking service should be able to access inference rule specifications dynamically and to use the specifications to check the proof steps as described and exemplified in Section 5.2

5.1 A PML Proof-Checking Service

We have used an InferenceML API to implement a proof-checking service and it has the following steps for matching:

1. Parse the rule specification into a parse tree
2. The internal nodes of this parse tree correspond to InferenceML operators like $+$, $-$, [] and the leaves correspond to the raw variables that occur in the InferenceML rule specification
3. Bindings for variables that stand for lists and singletons are provided by the proof that is being checked. Bindings for variables of numeric and substitution types are automatically computed by the system. For example, if the rule uses $A.i$ then all elements of the list A are tried automatically by the system and we do not need to explicitly specify the binding for the index variable i. A binding for the list variable A however, needs to be provided by the proof as described in the next section.

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/InfML.owl#InfML

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

name: "InferenceML – The Inference Meta Language"

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

name: "SCL – Simplified Common Logic"

Figure 1. IWBase proof meta-information.

4. The parse tree is then evaluated according to the bindings provided. So if we have a node with operator $+$ and its two children have lists A and B, then the nodes are collapsed into a single node that is a list of the union of the elements of A and B.

5. When the evaluation process is complete, we are left with a single root node and that is matched against the actual section from the nodeset file to see if everything matches.

The proof checker service has been implemented as a JSP application and a few demos are available at http://iw.stanford.edu/checker.html.

5.2 Proof Step Checking: An Example

Recalling the TonysSpecialty example in the introduction, 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. Node C is produced by the Wine Agent and it concludes that TonysSpecialty is a seafood dish. Node A is also produced by the Wine Agent deriving information provided by the food information extraction web service saying that TonysSpecialty is a crab dish (this part of the proof is not in Figure 2). Node B is from the food taxonomy service saying that crab is a subclass of seafood, So, each node set is represented as a stand-alone rectangle[3] 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 TonysSpecialty SEAFOOD)`, matches the rule conclusion, which is "`q[s]`";

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

¿From the `hasMetaBindings` property of the inference step, the checking service knows 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"). The service 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

[3]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 [17]. 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.

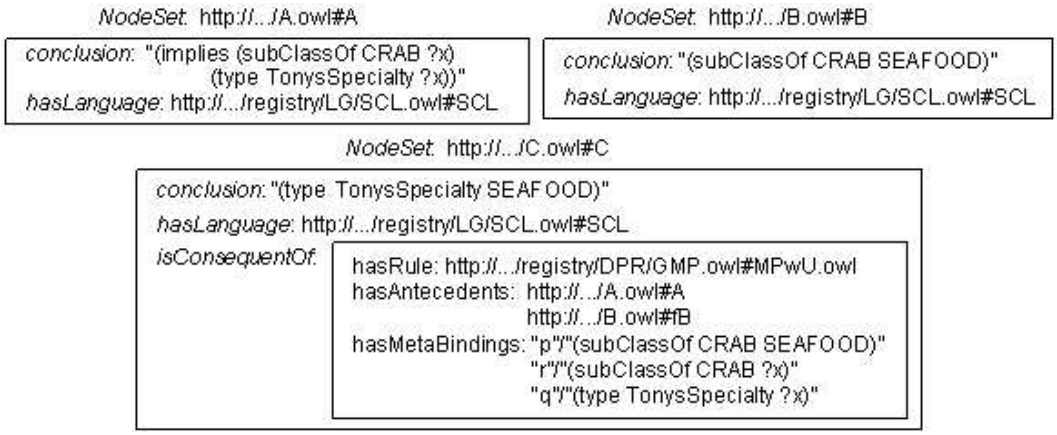

Figure 2. PML proof-level information.

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.

6 Related Work

Inference rules are often implemented rather than specified declaratively. Some systems, however, allow users to specify inference rules. For instance, Isabelle [16], as one of the many theorem provers that adopted Edinburgh LCF [8] 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 [19] 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 InferenceML 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 InferenceML, the OWL Rule Language (ORL)[9] 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 InferenceML is expected to be extended to incorporate some of the useful aspects of the rule specification languages described above. For example, we expect InferenceML 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, InferenceML 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 InferenceML.

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 [11, 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).

7 Conclusions

In this paper, we have described how to check hybrid proofs, which are proofs generated by multiple agents and services, each service using different sets of inference rules to derive their answers. We have also presented the need for a declarative language to annotate hybrid proofs with inference rule specifications. We also introduced the Inference Meta Language (InferenceML) as an option for such language. We eventually have described how hybrid proofs can be checked and provided an example.

InferenceML is a language that supports the specification of a wide variety of inference rules, enabling the checking of proofs in hybrid question answering systems. By providing a language for encoding inference rules, it facilitates proof checking (thereby improving checking and reliability of question answering systems). In addition, it supports proof combinations (thereby supporting interoperability), inference rule specification (thereby supporting justification access and presentation), and pattern specification (thereby supporting abstraction and matching).

InferenceML 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.

The work described here is functional but there are many desirable extensions yet to be made to the checker. Checking the validity of node set conclusions in languages other than SCL is planned future work as well as the extension of the InferenceML to check proofs for information extraction as described in [15].

The work and tools described in this paper including an InferenceML parser and checking service are available for use and feedback as described in the Inference Web website: http://iw.stanford.edu.

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] Lassaad Cheikhrouhou and Volker Sorge. PDS – A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence (ACIDCA'2000)*, Monastir, Tunisia, March 2000.

[6] Richard Fikes, Jessica Jenkins, and Gleb Frank. JTP: A System Architecture and Component Library for Hybrid Reasoning. Technical Report KSL-03-01, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, 2003.

[7] 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.

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

[9] 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.

[10] Eric I. Hsu and Deborah L. McGuinness. Wine Agent: Semantic Web Testbed Application. In *Proceedings of 2003 Description Logics (DL2003)*, Rome, Italy, September 5-7 2003. CEUR Workshop Proceedings.

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

[12] 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.

[13] 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.

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

[15] J. William Murdock, Paulo Pinheiro da Silva, David Ferrucci, Christopher Welty, and Deborah L. McGuinness. Encoding Extraction as Inferences. In *Proceedings of AAAI Spring Symposium on Metacognition on Computation*, Stanford University, USA, 2005. AAAI Press.

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

[17] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard Fikes. A Proof Markup Language for Semantic Web Services. *Information Systems*, 2004. (to appear).

[18] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Rob McCool. Knowledge Provenance Infrastructure. *IEEE Data Engineering Bulletin*, 25(2):179–227, December 2003.

[19] 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.

[20] Ilya Zaihrayeu, Paulo Pinheiro da Silva, and Deborah L. McGuinness. IWTrust: Improving User Trust in Answers from the Web. Technical Report DIT-04-086, Informatica e Telecomunicazioni, University of Trento, Trento, Italy, December 2004.