Explaining Subsumption in Description Logics

Deborah L. McGuinness
AT&T Bell Laboratories
Murray Hill, NJ 07974 U.S.A.
and
Dept. of Computer Science
Rutgers University
dlm@research.att.com

Alexander T. Borgida*
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903
U.S.A.
borgida@cs.rutgers.edu

Abstract

This paper explores the explanation of subsumption reasoning in Description Logics that are implemented using normalization methods, focusing on the perspective of knowledge engineers. The notion of explanation is *specified* using a proof-theoretic framework for presenting the inferences supported in these systems. The problem of overly long explanations is addressed by decomposing them into smaller, independent steps, using the notions of "atomic description" and "atomic justification". Implementation aspects are explored by considering the design space and some desiderata for explanation modules. This approach has been implemented for the CLASSIC knowledge representation system.

1 Introduction

Knowledge-based systems, like other software systems, need to be debugged while being developed. In addition, systems providing "expert advice" need to be able to justify their conclusions. Traditionally, developers have been supported during debugging by tools which offer a *trace* of the inferences performed by the system (e.g., a sequence of rule firings in a rule-based expert system), or, more generally by an explanation facility for the reasoner (e.g., [12]). Description Logics (DLs) form the basis of several recent knowledge-based systems, but do not currently offer such facilities. This is especially troublesome since DL reasoners perform a considerable variety of inference types, some of whose results have repeatedly proven, in practice, to be unexpected by developers.

Some DL-based Knowledge Base Management Systems (DL-KBMS), such as KRIS, are implemented using theorem proving techniques such as refutation-style tableaux techniques with rewrite rules [6]. Many other systems, such as LOOM, BACK and CLASSIC, are implemented using a Normalize-Compare approach, where implicit knowledge is first explicated into a normal form, after which subsumption can be checked quite quickly using simple so-called "structural" comparisons. This paper addresses the problem of explanation for this second class of DL-KBMS, which, although occasionally incomplete in terms of the inferences performed, tend to be more efficient for large and long-term KB maintenance.

An ad-hoc solution to the problem of explanation would be to take a particular implementation of a DL-KBMS and intersperse at appropriate places print-statements or other, more sophisticated operations, whose result could be assembled into an explanation. This approach is not only unprincipled but also unworkable, as we illustrate in Section 4.4 , because the declarative knowledge is compiled into data structures and imperative code in order to achieve efficiency. Our solution to this problem is to cast the various forms of DL reasoning into a single deductive framework based on rules of inference in the "natural deduction"-style,[1] and thence offer *proofs* as explanations of the system's conclusions.

However, even in simple cases "raw" proofs turn out to be too lengthy and non-modular. A second contribution of this work is a scheme for breaking up an explanation into steps, in a way which avoids superfluous details and which is incremental. This scheme is based on the idea of decomposing descriptions into "atomic" conjuncts, and providing "atomic justifications", which stand alone and may collapse several standard proof rules, or report only certain aspects of the proof rule application.

Finally, we discuss some of the issues and choices that arise in implementing such an explanation system as part of an existing Normalize-Compare reasoner. (In fact, we have implemented such a component for the CLASSIC KR&R system [4].) A particularly subtle concept constructor (**same-as**) is used to show how it is possible to integrate the theorem-proving aspect of searching for a proof/explanation, with a performance reasoner already available for the language.

2 Description Logics

Description Logics form a family of formalisms for representing and reasoning with knowledge, surveyed, among others, in [8; 16; 2]. The three fundamental notions of DLs are *individuals*, representing objects in the domain, *concepts*, describing sets of individuals, and *roles*, binary

*Supported in part by NSF Grant IRI-9119310.

[1] The rules of inference constitute a proof-theoretic semantics for the DL. Such semantics have been proposed for DLs in [1; 13].

Concept/Role	Denotation
THING	$\Delta^{\mathcal{I}}$
NOTHING	$\emptyset$
(**and** $\beta_1 \ldots \beta_n$)	$\beta_1^{\mathcal{I}} \cap \ldots \cap \beta_n^{\mathcal{I}}$
(**all** π β)	$\{d \in \Delta^{\mathcal{I}} \mid \pi^{\mathcal{I}}(d) \subseteq \beta^{\mathcal{I}}\}$
(**prim** $\alpha_1 \ldots \alpha_n$)	$\alpha_1^{\mathcal{I}} \cap \ldots \cap \alpha_n^{\mathcal{I}}$
(**at-least** n π)	$\{d \in \Delta^{\mathcal{I}} \mid \lvert\pi^{\mathcal{I}}(d)\rvert \geq n\}$
(**at-most** n π)	$\{d \in \Delta^{\mathcal{I}} \mid \lvert\pi^{\mathcal{I}}(d)\rvert \leq n\}$
(**same-as** π_1 π_2)	$\{d \in \Delta^{\mathcal{I}} \mid \pi_1^{\mathcal{I}}(d) = \pi_2^{\mathcal{I}}(d)\}$
(**fills** π $b_1,\ldots,b_m$)	$\{d \in \Delta^{\mathcal{I}} \mid b_i^{\mathcal{I}} \in \pi^{\mathcal{I}}(d)\}$
(**one-of** $b_1,\ldots,b_m$)	$\{b_1^{\mathcal{I}}, \ldots, b_m^{\mathcal{I}}\}$
identity	$\{(d,d) \mid d \in \Delta^{\mathcal{I}}\}$
(**inverse** π)	$\{(d,d') \mid (d',d) \in \pi^{\mathcal{I}}\}$

Table 1: Syntax and Denotational Semantics of Description Constructors (The symbols β, δ range over concepts, α ranges over atomic/primitive concept names, π over roles, b over individuals, and n over numbers.)

relations between individuals. Composite descriptions are formed using *constructors*, using a syntax such as the one in Table 1. In the following three examples:

```
RED-WINE ≡        (and (prim WINE) (fills color Red))
CABERNET ≡        (and (prim WINE) (fills grape
                   Cabernet) (fills color Red))
BLENDED-WINE ≡    (and (prim WINE) (at-least 2 grape)
                   (all grape WINE-GRAPE))
```

the concept RED-WINE is defined using the **and** constructor to conjoin the primitive concept WINE with the description of individuals whose color role is filled by the individual Red; CABERNET further constrains wine by specifying values for its color and grape roles.

The main deduction in DLs is *subsumption:* deciding whether one description, D1, is more general than another one, D2 (i.e., whether being a D2 logically implies being a D1). For example, CABERNET is subsumed by RED-WINE and is not subsumed by BLENDED-WINE. Most of the other kinds of inferences performed by DL-KBMS, such as detecting incoherent or disjoint concepts, and even individual recognition, can be represented using subsumption.

3 Explanation for Description Logics

An explanation is a justification of the system's beliefs. In rule-based and PROLOG systems, an explanation facility is often based on a trace of rule firings since rule invocation mimics modus ponens, which is the fundamental logical inference rule of the system. In DL-KBMS, an execution trace is inappropriate since the procedural implementation is too far removed from the underlying logic. For example, the implementation may traverse graph-like data structures to determine validity of coreference constraints, as illustrated in Section 4.4. This is similar to the problems faced by deductive databases such as LDL [11], where compilation for efficiency hides the original program's rule structure. As in LDL, explanation benefits from a declarative view of the reasoning, and we turn to the proof theory of description logics to obtain this. Proofs are built from logical axioms or

Ref $\quad \vdash \beta \Longrightarrow \beta$

Trans $\quad \dfrac{\vdash \beta \Longrightarrow \delta, \ \vdash \delta \Longrightarrow \lambda}{\vdash \beta \Longrightarrow \lambda}$

Eq $\quad \dfrac{\vdash \lambda \equiv \eta \qquad \vdash \beta \Longrightarrow \delta}{\vdash \beta\{\lambda/\eta\} \Longrightarrow \delta\{\lambda/\eta\}}$

Prim $\quad \dfrac{\tilde{\alpha} \subseteq \tilde{\theta}}{\vdash (\textbf{prim}\ \tilde{\theta}) \Longrightarrow (\textbf{prim}\ \tilde{\alpha})}$

THING $\quad \vdash \beta \Longrightarrow \text{THING}$

AndR $\quad \dfrac{\vdash \beta \Longrightarrow \delta, \ \vdash \beta \Longrightarrow (\textbf{and}\ \tilde{\lambda})}{\vdash \beta \Longrightarrow (\textbf{and}\ \delta\ \tilde{\lambda})}$

AndL $\quad \dfrac{\vdash \beta \Longrightarrow \delta}{\vdash (\textbf{and}\ \ldots\ \beta\ \ldots) \Longrightarrow \delta}$

All $\quad \dfrac{\vdash \beta \Longrightarrow \delta}{\vdash (\textbf{all}\ \pi\ \beta) \Longrightarrow (\textbf{all}\ \pi\ \delta)}$

AtLst $\quad \dfrac{n \geq m}{\vdash (\textbf{at-least}\ n\ \pi) \Longrightarrow (\textbf{at-least}\ m\ \pi)}$

AndEq $\quad \vdash \beta \equiv (\textbf{and}\ \beta)$

AtLs0 $\quad \vdash (\textbf{at-least}\ 0\ \pi) \equiv \text{THING}$

All-thing $\quad \vdash (\textbf{all}\ \pi\ \text{THING}) \equiv \text{THING}$

All-and $\quad \vdash (\textbf{and}\ (\textbf{all}\ \pi\ \beta\ )\ (\textbf{all}\ \pi\ \delta\ )\ \ldots\ ) \equiv (\textbf{and}\ (\textbf{all}\ \pi\ (\textbf{and}\ \beta\ \delta))\ \ldots)$

Figure 1: Subsumption and Equivalence Rules

told assumptions using rules of inference. Our rules are in the "natural semantics" style of [7], and show what new assertion can be considered "proven" on the basis of previously demonstrated ones. For example, *modus ponens* can be expressed as $\dfrac{\vdash \sigma \supset \rho \qquad \vdash \sigma}{\vdash \rho}$ indicating that if the two antecedent assertions, $\sigma \supset \rho$ and σ, can be deduced, then the consequent ρ is deducible in the next step of the proof. The subsumption rules of DLs can also be expressed in this form [1], by considering the subsumption judgment $\Delta \Longrightarrow \Gamma$. For example, the subsumption relationship between **all**-restrictions (namely, that the value restriction of the subsumed **all** restriction needs to be more specific than the value restriction of the subsuming **all** restriction) is expressed by the rule $\dfrac{\vdash \beta \Longrightarrow \delta}{\vdash (\textbf{all}\ \pi\ \beta) \Longrightarrow (\textbf{all}\ \pi\ \delta)}$. Similar inference rules can be obtained from the proof-theory based on the sequent calculus for DLs proposed in [13].

3.1 Explanations as Proofs

Since subsumption can be captured by inference rules, we can start by making the explanation of a subsumption relationship be its proof.

In order to provide an example, we will limit ourselves to a small DL, Mini-CLASSIC, which has one built-in concept, THING, and concept constructors **and**, **all**, **at-least**, and **prim**. Figure 1 provides the inference rules for determining subsumption.[2]

Consider a concept A, defined as

[2]Symbols with a superscript tilde, $\tilde{\alpha}$, indicate a set.

A ≡ (**and** (**at-least** 3 grape) (**prim** GOOD WINE))

and let us consider the proof of the subsumption

A ⟹ (**and** (**at-least** 2 grape) (**prim** WINE)).

We justify each line by the name of the rule used to deduce it, and the line numbers on which the antecedents, if any, of the rule appear:

1. (**at-least** 3 grape) ⟹ (**at-least** 2 grape) AtLst
2. (**and** (**at-least** 3 grape) (**prim** GOOD WINE)) ⟹ (**at-least** 2 grape) AndL,1
3. (**prim** GOOD WINE) ⟹ (**prim** WINE) Prim
4. (**and** (**at-least** 3 grape) (**prim** GOOD WINE)) ⟹ (**prim** WINE) AndL,3
5. A ≡ (**and** (**at-least** 3 grape) (**prim** GOOD WINE)) Told
6. A ⟹ (**prim** WINE) Eq,4,5
7. (**prim** WINE) ≡ (**and** (**prim** WINE)) AndEq
8. A ⟹ (**and** (**prim** WINE)) Eq,7,6
9. A ⟹ (**at-least** 2 grape) Eq,5,2
10. A ⟹ (**and** (**at-least** 2 grape) (**prim** WINE)) AndR,9,8

If such a proof is to be treated as an explanation, we must resolve several problems: First, the proof is much longer than one might have expected for such a simple case. It includes some lines that are obvious to most readers, e.g., applications of AndEq (creating an equivalent expression by adding an enclosing **and**) and Eq (creating an equivalent expression by substituting equivalent subexpressions). These should be avoided if we are to explain more complex DLs, which have many more rules. In the example above, the proof should contain just those steps that seem most critical: AtLst and Prim.

Second, we note that the proof has two major parts — one focusing on **at-least**, and one on the primitives. To deal with large, complex proofs, we propose to *decompose* them into parts that can be presented independently. This decomposition will be based on the use of AndR and AndL to break the concept into its component conjuncts, and then proceed with separate, smaller proofs of each part.

Finally, note that the current proof needs to be presented in its entirety since it contains inference rule applications that take proof line numbers as arguments. If proofs are very long, it might be more helpful to present individual steps of a proof that can stand alone. In other words, proof steps might take arguments that do not change according to the order of inference application, and do not draw their meaning by being a part of a particular proof. For example, a proof step might have the form

(**all** wines RED-WINE) ⟹ (**all** wines WINE)
because All(π=wines, β=RED-WINE, δ=WINE)

3.2 Explanations as Proof Fragments

According to Hempel and Oppenheim[5], the "basic pattern of scientific explanation" consists of (i) the *explanandum* — a sentence describing the phenomenon to be explained, and (ii) the *explanans* — the class of those sentences which are adduced to account for the phenomenon. The latter fall into two classes: one containing sentences which state certain antecedent conditions; the other is a set of sentences representing general laws.

Our explanandum is a subsumption judgement of the form $B \Longrightarrow C$, where B and C are descriptions. The two classes of explanans are the general laws (rules of inference) of the DL, and the antecedent conditions that are the arguments to these laws.

One way to prove that $B \Longrightarrow C$ is by the following two-step approach: *(1)* Find a description (**and** C_1 ... C_n) that is equivalent to C. *(2)* Show that $B \Longrightarrow C_i$ for i=1,...,n.

In this approach, the first step of an explanation for $B \Longrightarrow C$ offers a list of descriptions C_i, whose conjunction is equivalent to C, and then asks the user to specify one or more C_i for which $B \Longrightarrow C_i$ is to be explained.[3] In Normalize-Compare algorithms, of the kind we assume here, B is put into a normal form so that it is itself a conjunction (**and** B_1 ... B_m), and then $B \Longrightarrow C_i$ is proven by finding some component B_j such that $B_j \Longrightarrow C_i$. In this case, one must also explain how the conjuncts B_i of the normalized form were deduced.

Components like C_j should, ideally, be as simple as possible, so let us call them *atomic*. For example, a description, D_1, such as (**fills** *speaks Italian English*) should not be atomic, since it can be written as (**and** (**fills** *speaks Italian*) (**fills** *speaks English*)); each conjunct may be explained differently, and each such explanation would be one simpler piece of a complete explanation of D_1. The specification of a *general and absolute* criterion of atomicity is unfortunately not likely to succeed: If we ask for components D that do not have the property that $D \equiv$ (**and** E_1 E_2), then nothing is atomic, since every description satisfies $D \equiv$ (**and** D E) where E is more general than D. The notion of "reduced clauses" introduced in [15], which requires that the conjuncts E_i be distinct from D and THING, also fails to work, as shown by the following two examples:

NOTHING ≡ (**and** (**at-least** 2 p) (**at-most** 1 p))
(**one-of** 4) ≡ (**and** (**one-of** 2 4) (**one-of** 1 4))

Currently we distinguish "spurious" conjunctions (which should not be used as signs of non-atomicity) by the fact that they allow an unbounded number of distinct ways of rewriting the description D; e.g., in the above two cases one can use any integers k and l such that $4 < k < l$, not just 2 and 1.

Note that our general approach to explanation does not rely on any particular property of "atomic descriptions", and so the absence of a general definition for this notion is not a major problem. Moreover, we can not always avoid atomic justifications involving a form of conjunction: sometimes NOTHING is obtained from combining two conflicting bounds on some role, and this is an implicit conjunction of the bounds. As we shall see later, candidates for atomic descriptions should be found among the components of normalized concepts used in the implementation.

Once the subsuming concept is broken up into its atomic descriptions, we use *atomic justifications* to explain each subsumption relationship. (Note that when "atomizing" a description D into the form (**and** E_1 ...

[3] If a more automated approach is desired, the system can provide an explanation of all the C_i.

E_n), we should eliminate redundant conjuncts, i.e., E_i should not subsume E_j for $i \neq j$.) An atomic justification has the form

A $\Longrightarrow$ B **because** *ruleId(<argument list>)*.

where B is an atomic description, *ruleId* is the name of an inference rule, and *argument list* is a set of bindings for variables in the inference rule.

The simplest atomic justification is *told information*: a relationship holds because it was explicitly asserted by the KB builder, usually as part of a definition:

A $\Longrightarrow$ (**and** (**prim** GOOD WINE) (**at-least** 3 grape))
because told-info

We also report told-info as the reason why A is subsumed by each of the syntactically occurring conjuncts of its definition:

A $\Longrightarrow$ (**at-least** 3 **grape**) **because** told-info.

Concept definitions are expanded by the "inheritance" inference: If a concept B is defined as (**and** A ...), then one reason why Bs have at least 3 **grapes** is because of transitive inheritance through A:

B $\Longrightarrow$ (**at-least** 3 grape) **because** inheritance(δ=A).

Suppose we are given an additional concept C, having B as a conjunct; then C $\Longrightarrow$ B $\Longrightarrow$ A, so C would have (**at-least 3 grape**) by transitivity and inheritance through both B and A, whether or not B stated anything explicitly about the **grape** role. We may therefore limit inheritance to report only from the concept that contains the description as "told information":

C $\Longrightarrow$ (**at-least** 3 grape) **because** inheritance(δ=A).

Another atomic justification is based on the All rule:

(**all wines RED-WINE**) $\Longrightarrow$ (**all wines WINE**)
because All(π=**wines**, β=**RED-WINE**, δ=**WINE**).

Observe that this is not a complete explanation, since it leaves us with an intermediate explanandum: why was RED-WINE subsumed by WINE? To answer this, the user may request a separate atomic justification or may enter another mode where the system will automatically ask all appropriate follow-up questions. From the above example we can see that atomic justifications naturally *chain* backward, until one stops at (i) inference rules without antecedents (e.g., told information), (ii) rules whose antecedents are theorems of mathematics (e.g., AtLst), (iii) user-specified rules that are deemed unnecessary or stopping points (e.g., in CLASSIC explanation chains can be stopped when a user-defined rule fires).

There may be cases where there are multiple justifications (e.g., inheritance from several ancestors). For this, one might allow multiple (disjunctively branching) explanation chains. Of course, if an inference rule has multiple antecedents, the explanation "chain" needs to branch conjunctively. Because it is possible for explanation chains to become long and branching, we provide user control of chaining.

To conclude, we review the example from Section 3.1. Explaining A $\Longrightarrow$ (**and** (**at-least 2 grape**) (**prim WINE**)) is now equivalent to explaining why A is subsumed by each of the two atomic descriptions of the subsumer:

A $\Longrightarrow$ (**prim WINE**)
because Prim($\tilde{\theta}$={Wine,Good}, $\tilde{\alpha}$={Wine})
A $\Longrightarrow$ (**at-least 2 grape**)
because AtLst(n=3,m=2,π=**grape**)

For readability, in applications such as [10] we associate templates with the various inference rules, so that the above justification is actually reported as

"AtLeast-Ordering" on role **grape**: 3 is greater than 2.

Our work on such such "surface" presentation is still preliminary.

4 Developing an Explanation System

In order to build an explanation system based on the preceding theory, we need to accomplish the following tasks:

- Identify a (sub)language of atomic descriptions.
- Present "atomization" rules for normalizing a description into atomic conjuncts.
- Identify rules of inference for subsumption.
- Develop algorithms for (re-)constructing subsumption proofs.

4.1 Atomic Descriptions and Atomization

To define the grammar of atomic descriptions, one may begin with the grammar of the concept language, and eliminate those constructs that can be written as conjunctions of other, more general descriptions. Such constructs might be signaled by the presence of inference rules involving **and**, or having the form

$$\frac{\vdash C \Longrightarrow \ldots \qquad \vdash C \Longrightarrow \ldots}{C \Longrightarrow \ldots}$$

The following is a grammar of atomic descriptions for Mini-CLASSIC:

<atomic-descr> ::= THING |
(**at-least** <integer> <role-name>) |
(**prim** <identifier>) |
(**all** <role-name> <atomic-descr>)

One should verify that this grammar has the property that every description is equivalent to the conjunction of some set of atomic descriptions. If such a proof is constructive, it will usually identify a subset of inference rules that are needed for converting to normal form (see [9]).

For DL-KBMS using a Normalize-Compare algorithm for subsumption, the normal form of a concept can be mapped onto a set of atomic descriptions relatively easily, and hence forms a good basis for the atomization process. For example, CLASSIC's normal form is not quite atomic because it contains nested conjunctions, as in (**all r (and C D)**), and implicit conjunctions such as (**fills r** b_1 b_2). However, a simple routine can be written to break apart such nested conjunctions to yield atomic descriptions.

The following is the grammar of additional atomic descriptions needed for full CLASSIC:

<atomic-descr> ::= {CLASSIC|HOST}-THING |
(**at-most** <integer> <role-name>) |
(**one-of** <ind-name>*) |
(**test**{c|h} <test-name>) |
(**fills** <role-name> <ind-name>) |
(**same-as** (<att-name>+) (<att-name>+)) |
(**min** <number>) |
(**max** <number>)

4.2 Finding Subsumption Rules and Atomic Justifications

An explanation designer needs to identify the inference rules and the appropriate arguments that will be *reported* in explanations. It is wise to begin with a complete set of the inference rules required to derive all subsumptions. Royer and Quantz [13] describe an interesting systematic technique for obtaining inference rules for a DL from its translation to first order logic. One can also analyze the DL-KBMS implementation, looking for all updates to data structures, and expressing these as inference rules.

In order to control the verbosity of explanations, the developer can choose to limit the set of inferences that will actually be *reported* in explanations. This may mean skipping certain inferences, or merging two or more inferences together for the purposes of reporting. In CLASSIC, for example, we reduced the set of inference rules from the 100 or so original ones, to about half that number used in (the default settings for) our explanations. We also gave the user the option to add or delete some inferences to the set that will be reported.

In reporting an atomic justification, the implementer may choose to eliminate arguments that appear obvious. In CLASSIC we generally do not report arguments that appear in the denominator of the inference rule. For example, the Prim rule is reported only by name since all of its arguments appear in the denominator. Also, we do not report arguments like **at-least** values that can only have one value, but we do report which of the many potential parent concepts from which something was inherited. More details on filtering given context appear in [9].

4.3 The Explanation Construction Process

Typically, it requires too much space and effort to maintain enough information to allow a reconstruction of a formal subsumption proof for every deduction in a DL-KBMS. Thus, we expect that any extensive explanation facility will need to reconstruct proofs on demand, by "replaying" the normalization and subsumption algorithms. To do this, we need to choose between augmenting the core system code or writing separate explanation modules. In a tradeoff between impact on the core system and extra code that must be maintained, we agree with [14] and choose to minimize dual algorithm development. Thus, we have instrumented the main code to store more information when it is *replayed in explanation mode.*

Our implementation tries to explain a deduction from the current state information. If that is impossible, the system destroys certain derived information, and reruns the deductive part of the core system. This deductive core has been modified to include a rerun mode that calls an explanation update function whenever the system applies an inference rule that should be reported. The explanation update functions build structures that record the proof structure, and it is from these structures that the final explanations are generated. The only modifications to the core system are an extra pointer in the main data structures (for the explanation structure) and the rerun mode calls to explanation functions.

4.4 Explaining Co-reference Constraints

It will not always be possible to calculate explanations using the code implementing the DL reasoner. Consider the class of persons who share their name with both their parents, which is defined using the **same-as** constructor:

(**and** (**same-as** (mom name) (name))
(**same-as** (dad name) (name)))

This description is equivalent to:

(**and** (**same-as** (dad name) (mom name))
(**same-as** (dad name) (name)))

The canonical form for **same-as** involves a graph data structure [3], in which subsumption by a description of the form `(same-as (pp) (rr))` is checked by tracing the paths `pp` and `rr` to verify that they end at the same node. Unfortunately, the two concepts above may generate the same graph (in the same way that different regular expressions may generate the same finite automaton), though we would expect different explanations of why `(same-as (mom name) (dad name))` subsumes them: in the second case it is told information, while in the first case there are inferences to be made.

It is possible to encode the subsumption inferences for **same-as** as a logic program for predicate `eq(R,S)`, which, given certain "told equalities", finds all pairs of role paths R and S whose equality follows from them[4]. The rules include:

```
told-eq([dad,name], [name]).
base-eq(X,Y) :-told-eq(X,Y).
eq(X,X).
eq(X,Y) :- base-eq(X,Y).
eq(X,Y) :- append(Xa,Xb,X), base-eq(Xa,Va),
           append(Va,Xb,V),eq(V,Y).
```

where a description such as `(all p (same-as q r))` is encoded as the term `base-eq([p,q],[p,r])`.

Readers familiar with Prolog will realize that this program will easily enter an infinite loop. This can be avoided by forcing a breadth-first search of the goal tree, or equivalently, by putting a bound on the depth to be explored. This latter approach is feasible if we can bound a priori the number of inference rule applications. Although this could be done on purely theoretical grounds, we can obtain a much more accurate bound by considering the implementation data structure: if `(same-as (pp) (qq))` holds on some concept graph G, then only those equalities are potentially relevant which end up pointing to nodes on the paths traced from the root by following paths labeled by `pp` and `qq`. Therefore, by counting the in-degree (minus 1) of every node in the set traversed when following paths `pp` and `qq` (the initial node is not counted, and the final, common, node is only counted once) an upper bound on the depth to which to search is obtained. In other words, the maximal depth to be given for any particular search can be computed from the existing implementation structure, and then passed to the explanation component; in turn,

[4]Because of the added overhead of adding this algorithm and Prolog to our system, our current implementation offers a less detailed (and hence less accurate) explanation, which, in this case, produces the same result for both of the above examples.

this does an (inefficient, but bounded) exhaustive search of all possible inferences until a proof is found. This illustrates our point that the implemented description reasoner can be used as a source of hints for reconstructing detailed formal proofs, which can then be presented as explanations.

5 Conclusions

DL-KBMS require an explanation service in order to help knowledge engineers with debugging. In systems implemented using a normalize-compare algorithm (as opposed to a refutation technique), the task of providing such a service in a *principled way* is made difficult by the presence of procedural code manipulating data structures.

As a foundation for explanations of subsumption, we propose the use of a *deductive framework* based on proof rules in the "natural semantics" style. As part of this plan, we presented the rules of inference for a simple description logic, and showed how formal proofs of subsumptions are like explanations, albeit ones that are too onerous for users to read because of a surfeit of details. The advantage of the natural deduction style is that the resulting proofs have the familiar form of "backward chaining trees".

In order to simplify and break up explanations, we decompose concepts into atomic descriptions, for which subsumption is explained independently. Furthermore, rather than presenting complete proof trees, we explain individual nodes in the proof tree through atomic justifications, which make symbolic references to the antecedents, thereby allowing the user to choose if and when proofs for these should be presented.

Finally, we drew on our implementation and application experience to provide some general guidelines for developing explanation components for DLs. In particular, rather than developing an entirely separate "theorem prover" for finding proofs, we suggest augmenting the DL implementation with facilities for reconstructing proofs, when these are needed. The synergism between these two components was illustrated using the **same-as** constructor of CLASSIC.

We have only reported on subsumption explanation in a limited DL. Our full system explains *every* inference that CLASSIC makes, and also includes explanations of individual reasoning, errors detected, and why a concept does *not* subsume another object. We also provide extensive filtering methods based on meta-information for limiting presentations and explanations of objects.

Acknowledgments: We are indebted to Lori Alperin Resnick for her collaboration in implementing the explanation system for CLASSIC. We also gratefully acknowledge the enlightening comments on the work and its presentation offered by R. Brachman, W. Cohen, H. Hirsh, S. Hofmeister, H. Kautz, C. Kulikowski, P. Patel-Schneider, J. Moore, W. Swartout, R. Thomason, E. Weixelbaum, and J. Wright. Of course, all remaining errors are ours.

References

[1] A. Borgida. From Type Systems to Knowledge Representation: Natural Semantics Specifications for Description Logics. In *International Journal of Intelligent and Cooperative Information Systems*, pp.93–126, March 1992.

[2] A. Borgida. Description Logics in Data Management. To appear in *IEEE Trans. on Knowledge and Data Management*. An earlier version is available as Rutgers Tech. Report DCS-TR-295.

[3] A. Borgida and P. F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. In *Journal of Artificial Intelligence Research, vol. 1*, 1994, pp. 277–308.

[4] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *Principles of Semantic Networks: Explorations in the representation of knowledge*, J. Sowa, editor, Morgan-Kaufmann, 1991, pp. 401–456.

[5] C. G. Hempel and P. Oppenheim. Studies in the Logic of Explanation. In *The Structure of Scientific Thought*, E. H. Madden, editor, Riverside Press, 1960, pp. 19–29.

[6] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *Proc. 9th ECAI*, Stockholm, August 1990, pp.348–353.

[7] G. Kahn. Natural Semantics Rapport de Recherche No. 601, INRIA, Sophia Antipolis, France.

[8] R. M. MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1991.

[9] D. L. McGuinness and A. Borgida. Explaining Subsumption in Description Logics. Technical Report LCSR-TR-228, Dept. of Computer Science, Rutgers University, September 1994.

[10] D. L. McGuinness and L. Alperin Resnick. Description Logic in Practice: A CLASSIC Application. In *Proc. IJCAI*, Montreal, August 1995.

[11] O. Shmueli and S. Tsur. Logical Diagnosis of LDL Programs. In *New Generation Computing*, OHMSHA, LTD and Springer-Verlag, Volume 9, pp. 277–303, June 1991.

[12] R. Neches, W. R. Swartout and J. Moore. Explainable (and Maintainable) Expert Systems. In *Proc. IJCAI-85*, Los Angeles, CA., 1985, pp. 382–389.

[13] V. Royer and J. Quantz. Deriving inference rules for terminological logics. In *Logics in AI, Proc. of JELIA '92*, D. Pearce, G.Wegner (eds), Springer Verlag, 1992, pp.84–105.

[14] W. R. Swartout and J. D. Moore. Explanation in Second Generation Expert Systems. In Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors, *Second Generation Expert Systems*. Springer-Verlag, in Press.

[15] G. Teege. Making the Difference: A Subtraction Operation for Description Logics. In *Proc. KR-94*, Bonn, Germany. May, 1994. pp. 540–550.

[16] W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers and Mathematics With Applications*. 23(2-5), March 1992.