Explaining $\mathcal{ALC}$ Subsumption

Alex Borgida,[1] **Enrico Franconi** and **Ian Horrocks**[2]

Abstract. Knowledge representation systems, including ones based on Description Logics (DLs), use explanation facilities to, among others, debug knowledge bases. Until now, such facilities were not available for expressive DLs, whose reasoning is an un-natural refutation-based tableau. We offer a solution based on a sequent calculus that is closely related to the tableau implementation, exploiting its optimisations. The resulting proofs are pruned and then presented as simply as possible using templates.

1 Introduction

The usability of knowledge representation systems, including ones based on Description Logics (DLs) is considerably enhanced by the ability to explain inferences to knowledge-base developers who are not familiar with the implementation of the reasoner [10]. For DLs, inferring subsumption relationships is a fundamental reasoning task, and its explanation is relatively natural for systems based on structural subsumption algorithms [10]. However, such algorithms are unable to deal with a more complex language such as $\mathcal{ALC}$. Tableaux-based systems, on the other hand, can deal with $\mathcal{ALC}$ (and much more complex languages), but the reasoning method does not lead to a natural explanation of subsumption inferences because it is based on a "refutation/unsatisfiability" approach; for example, it would probably not be useful to have the subsumption $\forall R.(C \sqcap D) \sqsubseteq \forall R.C$ explained by the fact that $(\forall R.(C \sqcap D)) \sqcap \exists R.\neg C$ is not satisfiable.

This problem is not restricted to DL-based systems: in other areas of theorem proving, there is a desire to provide explanations of why theorems hold, yet the proof techniques (e.g., resolution) are not "natural". The solution in such situations [9, 7] has been to find ways to transform proofs from their original form into some more "natural" form, such as *natural deduction* (ND) proofs. From the beginning, ND proofs have been claimed to be easier to present to users, and natural language generation systems have even been built to produce sophisticated English output from ND proofs (e.g., VERBMOBIL project [8]). Related to ND proof systems are *sequent calculi*, introduced by Gentzen. Such calculi axiomatise the entailment relation, which has an obvious parallel with the subsumption relation.

However, most sequent calculi include reasoning rules and notation that are less than natural in the case of DLs, such as moving formulæ from one side of the turnstile to the other. Just as importantly, it is undesirable to have an explanation component that is dissociated from the implementation of the reasoner (the tableaux technique, in our case). This is both because of efficiency and the possible deviation between implementation and explanation [14, 10].

We propose using a slightly extended tableaux algorithm that, by keeping track of the "undesirable" steps involved in both the reduction of the subsumption problem to an unsatisfiability problem allows the structure of the original subsumption inference to be preserved (Section 2). Interestingly, the tableaux proof can be represented as a sequence of rules in a simple variant of the sequent calculus. Each rule application in this sequent calculus can then be explained in terms of one or more steps – some optional steps are omitted whenever possible, to produce a simpler proof – resulting in a parsimonious yet understandable proof presentation (Section 3).

2 Subsumption Proofs with Tableaux

In order to solve a *subsumption* problem using a tableaux-based procedure – which solves unsatisfiability problems – the subsumption has to be reduced to an equivalent satisfiability problem. That is, if we want to check that a concept expression C is subsumed by a concept expression D, we should check whether the concept $C \sqcap \neg D$ is not satisfiable, since $C \sqsubseteq D$ iff $C \sqcap \neg D \sqsubseteq \bot$. For example, in order to prove that (1)

$$(\exists \texttt{friend}.\top \sqcap \forall \texttt{friend}.\neg(\exists \texttt{child}.\neg \texttt{Doctor} \sqcup \exists \texttt{child}.\texttt{Lawyer})) \sqsubseteq (\exists \texttt{friend}.(\forall \texttt{child}.(\texttt{Rich} \sqcup \texttt{Doctor}))),$$

it is necessary to prove that the following concept does not have any model:

$$\exists \texttt{friend}.\top \sqcap$$
$$\forall \texttt{friend}.\neg((\exists \texttt{child}.\neg \texttt{Doctor}) \sqcup (\exists \texttt{child}.\texttt{Lawyer})) \sqcap$$
$$\neg\exists \texttt{friend}.(\forall \texttt{child}.(\texttt{Rich} \sqcup \texttt{Doctor})).$$

Moreover, tableaux algorithms typically transform the resulting concept into negation normal form using a combination of deMorgan's rules and modal normalisations (e.g., $\neg\exists R.C$ iff $\forall R.\neg C$) [4]. For example, the above concept would be immediately transformed into the following concept, which should be then checked for unsatisfiability:

$$\exists \texttt{friend}.\top \sqcap$$
$$\forall \texttt{friend}.((\forall \texttt{child}.\texttt{Doctor}) \sqcap (\forall \texttt{child}.\neg \texttt{Lawyer})) \sqcap$$
$$\forall \texttt{friend}.(\exists \texttt{child}.(\neg \texttt{Rich} \sqcap \neg \texttt{Doctor})).$$

Thus, the structure of the original problem is completely lost, and the explanation of the proof steps generated by the tableaux procedure wouldn't be understandable by the user. These problems can be overcome using a combination of *lazy unfolding* and *tagging*.

Lazy unfolding is an optimisation technique, widely used in implemented systems, that has the effect of delaying the normalisation of compound concepts until it is required by the progress of the tableaux expansion [5]. In the generation of the proof, the combination of a normalisation and a subsequent expansion rule corresponds to a single proof step. For example, normalising $\neg(a \sqcap b)$ to $\neg a \sqcup \neg b$, followed by an application of the tableaux $\sqcup$-*rule*, would be seen as a single proof step explained by some sort of $\neg\sqcap$-*rule*.

Secondly, by tagging the subsumer concept D during the initial transformation of the subsumption problem $C \sqsubseteq D$ into the satisfiability problem $C \sqcap \neg D^{\dagger}$, where † indicates the tagged concept, and by consistently tagging all concepts derived from it by applications of tableaux rules, it is always possible to determine whether a concept in a particular stage of the tableaux was derived from the subsumer, i.e., its negation plays the role of subsumer in the explanation step.

We will assume an unlabelled (often called *trace based* in the DL literature) tableaux based procedure for $\mathcal{ALC}$ [4], modified with the

[1] Dept. of Computer Science, Rutgers University, USA
borgida@cs.rutgers.edu
[2] Dept. of Computer Science, University of Manchester, UK
{franconi|horrocks}@cs.man.ac.uk

$$
\begin{array}{llcr}
(=) & X\,,\,a \;\vdash\; a\,,\,Y & & \\
(l\uparrow) & X\,,\,a\,,\,\neg a \;\vdash\; Y & X \;\vdash\; a\,,\,\neg a\,,\,Y & (r\uparrow) \\
(l\bot) & X\,,\,\bot \;\vdash\; Y & X \;\vdash\; \top\,,\,Y & (r\bot) \\
(l\wedge) & \dfrac{X\,,\,a\,,\,b \;\vdash\; Y}{X\,,\,a\sqcap b \;\vdash\; Y} & \dfrac{X \;\vdash\; a\,,\,Y \qquad X \;\vdash\; b\,,\,Y}{X \;\vdash\; a\sqcap b\,,\,Y} & (r\wedge) \\
(l\neg\wedge) & \dfrac{X\,,\,\neg a \;\vdash\; Y \qquad X\,,\,\neg b \;\vdash\; Y}{X\,,\,\neg(a\sqcap b) \;\vdash\; Y} & \dfrac{X \;\vdash\; \neg a\,,\,\neg b\,,\,Y}{X \;\vdash\; \neg(a\sqcap b)\,,\,Y} & (r\neg\wedge) \\
(l\vee) & \dfrac{X\,,\,a \;\vdash\; Y \qquad X\,,\,b \;\vdash\; Y}{X\,,\,a\sqcup b \;\vdash\; Y} & \dfrac{X \;\vdash\; a\,,\,b\,,\,Y}{X \;\vdash\; a\sqcup b\,,\,Y} & (r\vee) \\
(l\neg\vee) & \dfrac{X\,,\,\neg a\,,\,\neg b \;\vdash\; Y}{X\,,\,\neg(a\sqcup b) \;\vdash\; Y} & \dfrac{X \;\vdash\; \neg a\,,\,Y \qquad X \;\vdash\; \neg b\,,\,Y}{X \;\vdash\; \neg(a\sqcup b)\,,\,Y} & (r\neg\vee) \\
(l\neg\neg) & \dfrac{X\,,\,a \;\vdash\; Y}{X\,,\,\neg\neg a \;\vdash\; Y} & \dfrac{X \;\vdash\; a\,,\,Y}{X \;\vdash\; \neg\neg a\,,\,Y} & (r\neg\neg) \\
(l\Diamond) & \dfrac{X'\,,\,b \;\vdash\; Y'}{X\,,\,\exists r.b \;\vdash\; Y} & \dfrac{X' \;\vdash\; b\,,\,Y'}{X \;\vdash\; \forall r.b\,,\,Y} & (r\Box) \\
(l\neg\Box) & \dfrac{X'\,,\,\neg b \;\vdash\; Y'}{X\,,\,\neg\forall r.b \;\vdash\; Y} & \dfrac{X' \;\vdash\; \neg b\,,\,Y'}{X \;\vdash\; \neg\exists r.b\,,\,Y} & (r\neg\Diamond)
\end{array}
$$

where $X' = \{a \mid \forall r.a \in X\} \cup \{\neg a \mid \neg\exists r.a \in X\}$, *and*
$Y' = \{a \mid \exists r.a \in Y\} \cup \{\neg a \mid \neg\forall r.a \in Y\}$

Figure 1. Rules for $\mathcal{ALC}$

addition of lazy unfolding and tagging. As a notation for the proof generated by the modified tableaux procedure, we introduce here a simple sequent calculus. Sequent calculi for $\mathcal{ALC}$ can be obtained from the modal logic literature [3] by exploiting the correspondence between $\mathcal{ALC}$ and the multi-modal propositional logic $K_{(m)}$, with the subsumption relation being encoded as the entailment relation in a sequent. Note that there have already been attempts (e.g., [13]) to produce sequent calculi for DLs based on such calculi from predicate logic. However, the result does not bear a direct relation to our tableaux proof system, nor has it been used to generate short explanations. In order to devise the sequent notation, we exploit the well known fact that in classical logic it is possible to obtain a sequent proof directly from a standard tableaux satisfiability algorithm, where applications of tableaux rules correspond with steps in the sequent proof, and clash detections correspond with termination axioms (e.g., [12]).

The calculus is shown in Figure 1. Please note that the proposed system is not strictly original (see, e.g., [2]); what is important here is the way a sequent proof can be correlated with a tableaux procedure for $\mathcal{ALC}$, since all the implemented systems for expressive Description Logics make use of tableaux procedures. In order to parallel the behaviour introduced by lazy unfolding and tagging in the tableaux calculus, weakening and negation rules do not exist. If a negation rule is used in an explanation of subsumption, this would result in shifts of subsumers to subsumees and vice versa. On the other hand, new rules are introduced which explicitly consider negation in front of every construct. In order to parallel the behaviour of the $\forall$- and $\exists$-*rules* in the tableaux calculus, the applicability condition of the $\Box$- and $\Diamond$-*rules* is explicitly considered. The condition states that the rule is applicable if all the homologous universal and existential formulæ are "gathered" together on the left and right hand sides of the sequent in the precondition; the rule is then applied only once. Of course, additional termination axioms are also given.

We will now see how the tableaux algorithm would demonstrate the above-mentioned subsumption (1).

We want to parallel the steps in the tableaux algorithm with the corresponding sequent steps for the same proof (Figure 2). The sequent notation will be used in the next section to devise the explanation. The proof starts by proving the unsatisfiability of the set of concepts

$$\{(\exists\mathtt{friend}.\top \sqcap \forall\mathtt{friend}.\neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer}))),\ \neg(\exists\mathtt{friend}.(\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})))^{\dagger}\},$$

An application of the tableaux $\sqcap$-*rule* to the first concept leads to the set

$$\{\exists\mathtt{friend}.\top, \forall\mathtt{friend}.\neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer})),\ \neg(\exists\mathtt{friend}.(\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})))^{\dagger}\}.$$

Because the concept triggering the tableaux rule was not tagged, this corresponds to a sequent step using the $(l\wedge)$ rule (step 2 in Figure 2). The tableaux algorithm would then normalise

$$\neg(\exists\mathtt{friend}.(\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})))^{\dagger}$$

to give $\forall\mathtt{friend}.(\neg\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor}))^{\dagger}$, and apply the $\exists$-*rule* to $\exists\mathtt{friend}.\top$, generating the sub-problem consisting of

$$\{\top, \neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer})),\ \neg\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}\},$$

where $\neg\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}$ is tagged because it was derived from a tagged concept. Because the triggering concept was not tagged, this corresponds to a sequent step using the $(l\Diamond)$ rule (step 3 in Figure 2).

The next step in the tableaux algorithm would be a normalisation of $\neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer}))$ to give $\neg(\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcap \neg(\exists\mathtt{child}.\mathtt{Lawyer})$, followed by an application of the $\sqcap$-*rule*, leading to the set

$$\{\top, \neg(\exists\mathtt{child}.\neg\mathtt{Doctor}), \neg(\exists\mathtt{child}.\mathtt{Lawyer}),\ \neg\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}\}.$$

Because the triggering concept was not tagged, these two steps correspond to a sequent step using the $(l\neg\vee)$ rule (step 4 in Figure 2). The tableaux algorithm would then normalise all the negated concepts to give $\forall\mathtt{child}.\neg\neg\mathtt{Doctor}$, $\forall\mathtt{child}.\neg\mathtt{Lawyer}$ and $\exists\mathtt{child}.\neg(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}$ respectively. The $\exists$-*rule* would then be applied to the last of these concepts, generating

$$\{\neg\neg\mathtt{Doctor}, \neg\mathtt{Lawyer}, \neg(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}\}.$$

Because the triggering concept was tagged, this expansion corresponds to one of the sequent right rules, and the preceding normalisation step means that it corresponds to a sequent step using the $(r\Box)$ rule (step 5 in Figure 2).

The tableaux algorithm would then proceed with a normalisation of $\neg\neg\mathtt{Doctor}$, corresponding to a sequent step using the $(l\neg\neg)$ rule (step 6 in Figure 2), and a normalisation of $\neg(\mathtt{Rich} \sqcup \mathtt{Doctor})^{\dagger}$, followed by an application of the $\sqcap$-*rule* to give

$$\{\mathtt{Doctor}, \neg\mathtt{Lawyer}, \neg\mathtt{Rich}^{\dagger}, \neg\mathtt{Doctor}^{\dagger}\},$$

where $\neg\mathtt{Rich}$ and $\neg\mathtt{Doctor}$ are both tagged. The combination of the triggering tagged concept and the normalisation step means that this last expansion corresponds to the $(r\vee)$ sequent rule.

Finally, the tableau algorithm detects a clash between $\mathtt{Doctor}$ and $\neg\mathtt{Doctor}^{\dagger}$. Because $\neg\mathtt{Doctor}^{\dagger}$ is tagged, this corresponds to the sequent termination axiom $\mathtt{X}\,,\ \mathtt{Doctor} \;\vdash\; \mathtt{Doctor}\,,\ \mathtt{Y}$ (step 7 in Figure 2).

3 The surface structure of explanations

The sequent calculus proofs obtained from the theorem prover provide the framework for an explaination, but there are some problems with these proofs that would limit their usefulness as explanations to be given to end-users.

One problem relates to the contents of the proof tree itself: there can be several fragments of proofs which are irrelevant and would only clutter the exlanation. These can occur because of manipulations applied to concept fragments that end up being irrelevant. For example, in showing that $\neg\neg A \sqcap \forall r.C \sqcap \exists r.D$ is subsumed by $\exists r.(D \sqcup E)$, it is unnecessary to apply rule $(l\neg\neg)$, nor is it useful to carry the concept C, when applying rule $(l\Diamond)$. The solution is to simplify the sequent proof using a recursive analysis of the relevance of each component. This procedure has similarities with non-modal proof condensation techniques used in theorem proving (e.g., see [11]).

$$\begin{array}{lcr}
 & \exists\mathtt{friend}.\top \sqcap \forall\mathtt{friend}.\neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer})) & \\
 & \vdash\ \exists\mathtt{friend}.\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor}) & \\
(l\wedge) & \overline{\hspace{20em}} & (1) \\
 & \exists\mathtt{friend}.\top,\ \forall\mathtt{friend}.\neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer})) & \\
 & \vdash\ \exists\mathtt{friend}.\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor}) & \\
(l\Diamond) & \overline{\hspace{20em}} & (2) \\
 & \top,\ \neg((\exists\mathtt{child}.\neg\mathtt{Doctor}) \sqcup (\exists\mathtt{child}.\mathtt{Lawyer})) & \\
 & \vdash\ \forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor}) & \\
(l\neg\vee) & \overline{\hspace{20em}} & (3) \\
 & \top,\ \neg(\exists\mathtt{child}.\neg\mathtt{Doctor}),\ \neg(\exists\mathtt{child}.\mathtt{Lawyer}) & \\
 & \vdash\ \forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor}) & \\
(r\Box) & \overline{\hspace{20em}} & (4) \\
 & \neg\neg\mathtt{Doctor},\ \neg\mathtt{Lawyer}\ \vdash\ \mathtt{Rich} \sqcup \mathtt{Doctor} & \\
(l\neg\neg) & \overline{\hspace{20em}} & (5) \\
 & \mathtt{Doctor},\ \neg\mathtt{Lawyer}\ \vdash\ \mathtt{Rich} \sqcup \mathtt{Doctor} & \\
(r\vee) & \overline{\hspace{20em}} & (6) \\
 & \mathtt{Doctor},\ \neg\mathtt{Lawyer}\ \vdash\ \mathtt{Rich},\ \mathtt{Doctor} & \\
(=) & \overline{\hspace{20em}} & (7) \\
 & \text{TRUE} &
\end{array}$$

Figure 2. Sequent proof.

The other problems are related to the presentation of the sequent rules to users. The solution here mainly involves the use of templates to generate a surface explanation (in one or more steps) of each sequent rule application.

3.1 Simplifying proofs

The sequent proof tree found by the theorem prover will be assumed to be presented as a term, where the term constructor will be the rule name, and its arguments will include the important meta-variables appearing in the sequent rule, as well as any sub-proofs.

For example, consider a proof that starts with the application of the $(r\neg\sqcap)$ rule to

$$\neg\mathtt{Doctor}, \neg\neg\mathtt{Lawyer}\ \vdash\ \neg(\mathtt{Rich} \sqcap \mathtt{Doctor}),$$

leading to $\neg\mathtt{Doctor}, \neg\neg\mathtt{Lawyer}\ \vdash\ \neg\mathtt{Rich}, \neg\mathtt{Doctor}$, followed by an application of $(l\neg\neg)$, and then the termination axiom $(=)$ with $\neg\mathtt{Doctor}$. This would be encoded as the term

$$\mathsf{rNotAnd}(\mathtt{Rich}, \mathtt{Doctor}, \mathsf{lNotNot}(\mathtt{Lawyer}, \mathsf{ident}(\neg\mathtt{Doctor})))$$

where, for example, the type of proof constructor lNotNot is *Description*×*Proof*.

We now describe a function *Relevant*, which takes a proof and simplifies it so that only relevant proof steps are kept and, for modal rules, only those descriptions that are relevant to later parts of the proof are carried into the sub-proof. This is accomplished by computing two such sets of relevant terms (one for each side of $\vdash$) for *every* step of the proof. The function *Relevant* takes as argument a *Proof*, and returns a three-tuple: the revised proof, plus the above-mentioned two sets of concept terms, from which the current sequent can be reconstructed. The function is defined by case analysis of the proof step constructors, and is presented for some representative cases using pseudo-ML code, with pattern-matching notation.

Starting with the termination rules, we have, for example

```
Relevant( ident(A) ) = ( ident(A) , {A}, {A} )
Relevant( lBot() ) = ( lBot() , $\emptyset$, $\emptyset$)
Relevant( lContrad(A) ) = ( lContrad(A) , {A, $\neg$A},$\emptyset$)
```

The code for $\mathsf{lNotNot}(A, Pf)$ would first recursively process its sub-proof Pf (see the **let** statement in the following pseudo-code); if it turns out that the concept A was not needed (to detect a termination) in the sub-proof, then $\neg\neg A$ itself is irrelevant to the proof, and the step is skipped. Otherwise, the sub-proof explains where the A is used, and makes us be interested in explaining where the $\neg\neg A$ came from.

```
Relevant( lNotNot(A,Pf) ) =
  let (Pf1,Lhs1,Rhs1) = Relevant(Pf) in
    if (A $\in$ Lhs1)
    then (lNotNot(A,Pf1), Lhs1 $-$ {A} $\cup$ {$\neg\neg$A}, Rhs1 )
    else (Pf1,Lhs1,Rhs1)
```

Similarly, in dealing with $\mathsf{rNotAnd}(A, B, Pf)$, if neither $\neg A$ nor $\neg B$ was useful in the sub-proof then the rule can be skipped:

```
Relevant( rNotAnd(A,B,Pf) ) =
  let (Pf1,Lhs1,Rhs1) = Relevant(Pf) in
    if ($\neg$A $\in$ Rhs1) orelse ($\neg$B $\in$ Rhs1)
    then ( rNotAnd(A,B,Pf1), Lhs1,
           Rhs1 $-$ {$\neg$A,$\neg$B} $\cup$ {$\neg$(A$\sqcap$B)} )
    else (Pf1,Lhs1,Rhs1)
```

For example, by applying *Relevant* to proof 3.1, we would obtain the proof $\mathsf{rNotAnd}(\mathtt{Rich}, \mathtt{Doctor}, \mathsf{ident}(\neg\mathtt{Doctor}))$, plus the sets $\{\neg\mathtt{Doctor}\}$ and $\{\neg(\mathtt{Rich}\sqcap\mathtt{Doctor})\}$, representing the relevant parts of the sequent derived by the proof.

Finally, we need to consider the modal rules. First, observe that modal rule applications appearing in our proof can no longer be "useless", so our main task will be to thin out the terms that are carried into the sub-proof.

For example, in proving that

$$\begin{array}{l}\forall\mathtt{child}.\mathtt{Adult},\ \exists\mathtt{child}.\neg\neg\mathtt{Doctor} \\ \vdash\ \exists\mathtt{child}.\mathtt{Doctor},\ \exists\mathtt{child}.\mathtt{Rich}\end{array} \qquad (3.1)$$

rule $(l\Diamond)$ would gather *every* restriction on role $\mathtt{child}$, producing the subgoal $\mathtt{Person}, \neg\neg\mathtt{Doctor}\ \vdash\ \mathtt{Doctor}, \mathtt{Rich}$. However, the only relevant parts from both sides are those dealing with $\mathtt{Doctor}$, so we will want the sequent produced to be only $\exists\mathtt{child}.\neg\neg\mathtt{Doctor}\ \vdash\ \exists\mathtt{child}.\mathtt{Doctor}$.

When applying modal rules we need to track the precise form of the relevant subconcepts. We therefore distinguish in the antecedent X' of the top sequent those terms that come from formulæ of the form $\forall r.a$ from those that come from formulæ of the form $\neg\exists r.a$; likewise for the succedent Y'. Thus the $(l\Diamond)$ rule needs a list of four arguments: La, Lns, Rs and Rna, such that the antecedent $X' = La \cup Lns$ and the succedent $Y' = Rs \cup Rna$, where, for example, $La = \{a \mid \forall r.a\ \in X\}$ and $Lns = \{\neg a \mid\ \neg\exists r.a\ \in X\}$. Each of these sets may be diminished by the "relevance" list $Lhs1$ returned by the recursive call on the sub-proof:

```
Relevant( lSome(p,B,[La,Lns,Rs,Rna],Pf) ) =
  let (Pf1,Lhs1,Rhs1) = Relevant(Pf)  and
     (La1=La $\cap$ Lhs1) and (Lns1=Lns $\cap$ Lhs1) and
     (Rs1=Rs $\cap$ Rhs1) and (Rna1=Rna $\cap$ Rhs1)
  in (lSome(p,B,[La1,Lns1,Rs1,Rna1],Pf1) ,
     {$\exists$p.B}$\cup$ {$\forall$p.A | A $\in$ La1} $\cup$ {$\neg\exists$p.A | $\neg$A $\in$ Lns1},
     {$\exists$p.A | A $\in$ Rs1} $\cup$ {$\neg\forall$p.A | $\neg$A $\in$ Rna1}
      )
```

As a result, the sequent proof of (3.1), represented by

$$\begin{array}{l}\mathsf{lSome}(\mathtt{child},\neg\neg\mathtt{Doctor},\ [\{\mathtt{Adult}\},\emptyset,\{\mathtt{Doctor},\mathtt{Rich}\},\emptyset], \\ \qquad \mathsf{lNotNot}(\mathtt{Doctor},\mathsf{ident}(\mathtt{Doctor})))\end{array}$$

is reduced to

$$\begin{array}{l}\mathsf{lSome}(\mathtt{child},\neg\neg\mathtt{Doctor},\ [\emptyset,\emptyset,\{\mathtt{Doctor}\},\emptyset], \\ \qquad \mathsf{lNotNot}(\mathtt{Doctor},\mathsf{ident}(\mathtt{Doctor})))\end{array}$$

In the original example, from Section 2, *Relevant* eliminates $\neg\exists\mathtt{child}.\mathtt{Lawyer}$ from the application of $r\Box$, by replacing $\{\neg\neg\mathtt{Doctor},\neg\mathtt{Lawyer}\}$ with $\{\neg\neg\mathtt{Doctor}\}$ in the original proof step $\mathsf{rAll}(\mathtt{child}, (\mathtt{Rich} \sqcup \mathtt{Doctor}), [\emptyset, \{\neg\neg\mathtt{Doctor}, \neg\mathtt{Lawyer}\}, \emptyset, \emptyset], \_)$.

3.2 Generating surface explanations from pruned proofs.

The following problems arise when trying to offer the (reduced) sequent proof as an explanation:

1. The use of the comma as a separator on the antecedent side is semantically equivalent to conjunction, while on the succedent it is disjunction; this is quite confusing to non-initiates. On the other

$$\begin{array}{llll}
(\mathsf{NotNot}) & \neg\neg a \equiv a & & \\
(\mathsf{NotAnd}) & \neg(a \sqcap b \cdots) \equiv \neg a \sqcup \neg b \cdots & \neg(a \sqcup b \cdots) \equiv \neg a \sqcap \neg b \cdots & (\mathsf{NotOr}) \\
(\mathsf{NotSome}) & \neg\exists r.a \equiv \forall r.\neg a & \neg\forall r.a \equiv \exists r.\neg a & (\mathsf{NotAll}) \\
(\mathsf{AndAll}) & \forall r.a \sqcap \forall r.b \cdots \equiv \forall r.(a \sqcap b \cdots) & \exists r.a \sqcup \exists r.b \cdots \equiv \exists r.(a \sqcup b \cdots) & (\mathsf{OrSome}) \\
(\mathsf{SomeBot}) & \exists r.\bot \equiv \bot & \forall r.\top \equiv \top & (\mathsf{AllTop}) \\
(\mathsf{SomeAndAll}) & \exists r.a \sqcap \forall r.b \equiv \exists r.(a \sqcap b) \sqcap \forall r.b & &
\end{array}$$

Figure 3. Rules for concept equivalence used in explanations

hand, if we do not use the comma notation, inferences dealing with the commutativity and associativity of simple propositional connectives clutter explanations unnecessarily.

2. Several rules are identical on the left and right hand side, and might therefore be better presented as single rules for massaging concepts into an equivalent form (e.g., using de Morgan's rules).
3. The inference rules for modal formulae are quite complex and their validity is entirely non-obvious (and hence not a proper explanation step), since it is based on model-theoretic arguments inaccessible to naive users. This is in contrast to a structural subsumption rule such as $\frac{A \sqsubseteq B}{\forall r.A \sqsubseteq \forall r.B}$, which is self evident.
4. Proofs could, in general, be exponential in size. However, based on experience with CLASSIC, in explaining a subsumption of the form $A \sqsubseteq C \sqcap D$, experience with CLASSIC suggests that users often see one of the subsumptions ($A \sqsubseteq C$ or $A \sqsubseteq D$), and only want the other one explained. Since the problem is in PSPACE, this means that single branches are at most polynomial in size.

To resolve these problems we propose an approach based on the following idea: *Each proof step in the sequent calculus is expressed in terms of zero or more explanation rules (ERs) to be introduced, plus some choice on how to proceed with the rest of the explanation.*

To be clear, henceforth we will use $\sqsubseteq$ instead of $\vdash$ to indicate the subsumption relationships that are being explained, and refer to the antecedent and succedent as "lhs" and "rhs" of the subsumption.

First, we make conjunction and disjunction explicit on the lhs and rhs, replacing the commas. However, we will leave implicit all manipulations relating to associativity and commutativity of these operators. Therefore, sequent rules $(l\wedge)$ and $(r\vee)$ will not appear in the surface explanation.

Second, we introduce, in Figure 3, a variety of ERs that replace concepts by equivalent ones using, for example, the familiar rules of de Morgan. These rules can be applied to concepts in proof steps.

Each of these rules has an English template describing its application, and possibly a "because" clause, which the user may ask for in order to explain the rule itself. (This should be unnecessary, except for one rule, marked with * in Figure 4, which may well be treated as a lemma.)

For the modal rules, we offer simpler variants, which will be combined with equivalence rules (when necessary) to produce the same effect as the corresponding sequent rules. To begin with, $(l\neg\Box)$ and $(r\neg\Diamond)$ are explained as applications of de Morgan's law followed by $(l\Diamond)$ or $(r\Box)$ respectively. Then, the (AndAll) and (OrSome) equivalences can be used to gather together relevant components on the lhs and rhs. Finally, the subsumption can be explained using the (mostly) structural rules given in Figure 4.

We are now ready to sketch the proof explanation function *Explain*$(\cdot,\cdot)$, which, given the subsumption $\alpha \sqsubseteq \beta$ and its proof, generates some text, possibly offering further sub-explanation(s). Again, we consider a variety of sequent rule kinds to illustrate our approach.

For a termination sequent rule such as $(=)$, *Explain*$(\alpha \sqsubseteq \beta$, ident$(Z))$ would say: *"The subsumption now follows because the description Z is subsumed by Z, *and the lhs is a constriction of Z, since it is a conjunction, while **the rhs is an expansion of Z, since it is a disjunction"*. The sentence fragment starting at * (resp. **) is omitted if α (resp. β) is a singleton rather than a conjunct.

Next, consider an equivalence rule, like $(l\neg\neg)$. *Explain*$(\alpha \sqsubseteq \beta$, lNotNot$(Z, Pf))$ produces: *"Double negation elimination on the lhs leaves Z"*.

As promised, rules $(l\wedge)$ and $(r\vee)$ are not explicitly reported, so that *Explain*$(\alpha \sqsubseteq \beta$, lAnd$(A, B, Pf))$ just invokes *Explain*$(\alpha' \sqsubseteq \beta$, $Pf)$, where α' may have conjunction nesting removed from α. Rules $(l\vee)$ and $(r\wedge)$ represent case analysis, and offer the user a choice of which branch of the proof to follow (or stacks the proofs for both cases).

Finally, we come to modal rules. Let us consider

$$\textit{Explain}(\alpha \sqsubseteq \beta, \mathsf{lSome}(r, B, [La1, Lns1, Rs1, Rna1], Pf)).$$

If $Lns1$ is not empty, then we first apply equivalence rule (NotSome) to the elements of α which appear in $\{\neg\exists r.C \mid C \in Lns1\}$. Similarly for $Rna1$. This leaves us an explanation of the form *Explain*$(\alpha \sqsubseteq \beta$, lSome$(r, B, [La2, \emptyset, Rs2, \emptyset], Pf))$. If $La2$ is non-empty, then we apply equivalence rules (AndAll) (unless $La2$ is a singleton) and (SomeAndAll) to the subsumee to gather the $\forall$-restrictions and absorb them into $\exists r.B$. This leaves us with an explanation of the form *Explain*$(\alpha \sqsubseteq \beta$, lSome$(r, B, [\emptyset, \emptyset, Rs2, \emptyset], Pf))$. If $Rs2$ contains more than one element, they can be gathered into a single $\exists$-restriction using the (OrSome) equivalence rule. We now have two cases

- $Rs2$ is the empty set. In this case the subsumee must be incoherent, so we say: *"To prove that $\alpha \sqsubseteq \beta$, we will show that $\exists r.B$ is incoherent, and hence is subsumed by everything. For this, it is sufficient to show that B is incoherent."*. If we wanted, we could now introduce a variant of *Explain*$(\cdot,\cdot)$, call it ExplainIncoherent(B, Pf), which knows that the proof Pf only deals with the lhs, since it had an empty relevant rhs. Or we can continue with *Explain*$(B \sqsubseteq \bot, Pf)$.
- $Rs2$ contains a single concept C. In this case, we can use (StructSome) to provide a simple explanation: *"*The subsumption can now be proven by showing that $\exists$-restrictions for role r subsume. To prove $\exists r.X \sqsubseteq \exists r.Y$, it is sufficient to prove $X \sqsubseteq Y$. So in this case we are reduced to showing $B \sqsubseteq C$"*. The first sentence is omitted if α has no conjuncts.

Returning to our original example, we were asked to explain

$$(\exists\texttt{friend}.\top \sqcap \forall\texttt{friend}.\neg(\exists\texttt{child}.\neg\texttt{Doctor} \sqcup \exists\texttt{child}.\texttt{Lawyer})) \sqsubseteq (\exists\texttt{friend}.(\forall\texttt{child}.(\texttt{Rich} \sqcup \texttt{Doctor}))),$$

The (pruned) sequent proof is constructed with lAnd, lSome, lNotOr, rAll, lNotNot, rOr and ident. According to our rules, we skip lAnd, and explain lSome, which here uses ERs (SomeAndAll) and (StructSome). These generate the text

> *On the lhs, $\exists\texttt{friend}.\top$ can be strengthened with $\forall$-restrictions on role* `friend` *to yield $\exists\texttt{friend}.(\top \sqcap \neg((\exists\texttt{child}.\neg\texttt{Doctor}) \sqcup (\exists\texttt{child}.\texttt{Lawyer})))$. The subsumption can now be proven by showing that $\exists$-restrictions for role* `friend` *subsume. To prove $\exists\texttt{friend}.X \sqsubseteq \exists\texttt{friend}.Y$, it is sufficient to prove $X \sqsubseteq Y$. So in this case we are reduced to showing $\top \sqcap \neg((\exists\texttt{child}.\neg\texttt{Doctor}) \sqcup (\exists\texttt{child}.\texttt{Lawyer})) \sqsubseteq \forall\texttt{child}.(\texttt{Rich} \sqcup \texttt{Doctor})$.*

For rAll$(\texttt{child}, (\texttt{Rich} \sqcup \texttt{Doctor}), [\emptyset, \{\neg\texttt{Doctor}\}, \emptyset, \emptyset], _)$, we use ERs (NotSome) (combined with the preceding (NotOr), and then (StructAll) to get

> *On the lhs, apply de Morgan's laws to propagate in negation, to get $(\forall\texttt{child}.\neg\neg\texttt{Doctor})$. To prove $\forall\texttt{child}.X \sqsubseteq \forall\texttt{child}.Y$, it is sufficient to show $X \sqsubseteq Y$. So in this case we are reduced to showing $\neg\neg\texttt{Doctor} \sqsubseteq (\texttt{Rich} \sqcup \texttt{Doctor})$.*

(Note that $\neg\exists child.Lawyer$ had been pruned from the explanation.) From lNotNot we get

> *Double negation elimination on the lhs leaves* `Doctor`.

Finally, skipping rOr, ident produces:

> *The subsumption now follows because the description* `Doctor` *is subsumed by* `Doctor`*, and the rhs is an expansion of* `Doctor`*, since it is a disjunction.*

$$(\mathsf{StructSome})\ \frac{a\sqsubseteq b}{\exists r.a\sqsubseteq\exists r.b} \qquad \frac{a\sqsubseteq b}{\forall r.a\sqsubseteq\forall r.b}\ (\mathsf{StructAll})$$

$$\frac{a\sqsubseteq(b\sqcup c)}{\forall r.a\sqsubseteq(\forall r.b\sqcup\exists r.c)}\ (\mathsf{AllIsaAllSome}*)$$

Figure 4. ERs for role-restriction subsumptions

4 Extensions

Implemented DL systems support additional features such as terminological definitions (*Tbox*) and role hierarchies. Explanations should support these as well.

A Tbox is a set of axioms of the form $C \sqsubseteq D$ or $C \doteq D$ that assert subsumption relationships between (possibly complex) concepts. In many DLs, axioms are restricted to be "definitions": the left hand side is an atomic concept name that does not occur on the left hand side of any other axiom, and is not referred to (either directly or indirectly) in the right hand side. Examples of definition axioms are $\mathtt{Doctor} \sqsubseteq \mathtt{WellPaid}$ and $\mathtt{RichKids} \doteq \forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})$.

When the Tbox is restricted to definition axioms, reasoning involves simply expanding names with the associated Tbox definitions during a proof. For example, if the concept RichKids were defined as in the previous paragraph and used in our earlier example instead of $\forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})$, then it could be expanded to $\mathtt{RichKids} \sqcap \forall\mathtt{child}.(\mathtt{Rich} \sqcup \mathtt{Doctor})$ during the proof.[3] This procedure can be captured by simple sequent rules with side conditions, such as $\frac{c,d,X\ \vdash\ Y}{c,X\ \vdash\ Y}$ if $(c \doteq d) \in$ Tbox, and explained by a corresponding ER rules such as: *"α can be strengthened to $\alpha \sqcap \beta$ because $\alpha \doteq \beta$ is in the Tbox"*.[4] Moreover, correspondence with the tableaux algorithm is maintained because the "lazy unfolding" optimisation behaves in exactly the same way: defined concept names are only unfolded (expanded) as required by the progress of the tableaux proof. A very similar technique can be applied with necessary conditions on primitive concepts (i.e., when C is an atomic concept and $C \sqsubseteq D \in$ Tbox).

Extending the tableaux algorithm to deal with transitive roles and role hierarchies requires only a relatively minor extension to the $\exists$-*rule* [5], and a corresponding extension to the modal sequent rules is possible. (This extension leads to some problems with termination in the tableaux algorithm, but this is irrelevant for the explanation component, which receives a completed proof.)

Modern DL implementations also support a variety of of optimisation techniques. Some (like lazy unfolding) facilitate the generation of parsimonious explanations. For example, backjumping, a technique for pruning irrelevant search, complements the simplification procedure described in Section 3.1, while caching, a technique for reusing sub-proofs, can be used as a lemma generator.

Other extensions, such as general Tbox axioms and semantic branching optimisation complicate the task of explanation, and will be treated in future work.

5 Conclusions

Explanation of subsumption in expressive description logics was thought to be hard because the standard proof techniques are tableau-based, and are thus unnatural.

We have proposed a methodology for explaining the subsumption relationship between $\mathcal{ALC}$ concepts based on (i) a sequent proof for $\mathcal{ALC}$ derived from a tableaux based procedure, (ii) a pruning algorithm that eliminates unnecessary steps, and (iii) a set of templates used to generate "surface" explanations (in one or more steps) from each sequent rule application. The significant properties of this proposal include the fact that (a) the proof does not move terms from one side of the turnstile to the other, thus preserving the structure of the original subsumption, *yet can be obtained directly from a slightly enhanced tableaux algorithm*; (b) the surface explanation rules are used to provide the simplest explanation possible, avoiding in most cases the use of the one ER that, we believe, would itself require explanation (namely ($\mathsf{AllIsaAllSome}$)).

In some situations, our approach may in fact be better than one based on naive structural subsumption even for languages where the latter proof technique is applicable. For example, in showing that $C \sqcap \alpha \sqsubseteq C$, a structural subsumption technique first normalises the lhs, which is unnecessary in this case, yet might involve complex inferences if, for example, α contains an incoherence. The same would apply if C is a defined concept, when normalisation expands definitions, while our lazy unfolding would find the proof immediately.

The choice of the surface explanation rules, and especially their English language realization is to some extent provisional in the current work. This is particularly the case with proofs involving role-restrictions.

Although much more sophisticated natural language text generation is possible from proofs (e.g. [8]), we believe that for users building DL knowledge bases, a point-by-point explanation of the inferences, as done earlier in CLASSIC, is sufficient. We are aware that a considerable number of theorem provers have been endowed with the ability to produce explanations from the steps of proofs, and some of them, e.g., ILF [1], even offer a service whereby proofs obtained from seemingly arbitrary axiomatisations can submitted to obtain a surface English explanation, if the proof system can be reformulated appropriately. The very specialised nature of concept descriptions in DLs and modal logics (in contrast to FOL formulas) and of the subsumption proof itself, have led us to develop for now our own relatively simple surface generator, rather than trying to find a translation of our proofs into yet another form (e.g., ILF's "block proofs"), which may not be natural in the end.

We plan to concentrate instead on extensions to more expressive DLs and their highly optimised implementations, as well as the investigation of proof tactics that produce short or most easily explainable proofs.

Acknowledgements We wish to thank our original collaborators, Deborah McGuinness and Peter Patel-Schneider for their contributions, as well as very useful pointers offered by Matthew Stone, Hans Juergen Ohlbach, and by an anonymous referee. The first author was supported by NSF-grant IRI9619979.

REFERENCES

[1] B. I. Dahn, J. Gehne, T. Honigmann and A. Wolf, 'Integration of automated and interactive theorem proving in ILF', *Proc. of CADE-14*, pp. 57–60, (1997).

[2] S. Demri and R. Goré, 'Display calculi for logics with relative accessibility relations', *JoLLI*, (1999).

[3] M. Fitting, *Proof Methods for Modal and Intuitionistic Logics*, Kluwer, 1983.

[4] B. Hollunder and W. Nutt, 'Subsumption algorithms for concept languages', in *Proc. of ECAI'90*, pp. 348–353, (1990).

[5] I. Horrocks, 'Using an expressive description logic: FaCT or fiction?', in *Proc. of KR'98*, pp. 636–647, (1998).

[6] I. Horrocks and P. F. Patel-Schneider, 'Optimising description logic subsumption', *JLC*, **9**(3), 267–293, (1999).

[7] X. Huang, 'Reconstructing proofs at the assertion level', in *Proc. of CADE-94*, (1994).

[8] X. Huang and A. Fiedler, 'Presenting machine-found proofs', in *Proc. of CADE-96*, (1996).

[9] C. Lingenfelder, 'Transformation of refutation graphs into natural deduction proofs', SEKI-Rep. 86-10, Univ. of Kaiserslautern, (1986).

[10] D. McGuinness and A. Borgida, 'Explaining subsumption in description logics', in *Proc. of IJCAI'95*, pp. 816–821, (1995).

[11] F. Oppacher and E. Suen, 'HARP: A tableau-based theorem prover', *Journal of Automated Reasoning*, **4**, 69–100, (1988).

[12] S. Reeves and M. Clarke, *Logic for Computer Science*, Addison Wesley, 1993.

[13] V. Royer and J. Quantz, 'Deriving inference rules for terminological logics', in *Proc. of JELIA'92*, pp. 84–105, (1992).

[14] W. R. Swartout and J. D. Moore, 'Explanation in second generation expert systems', in *Second Generation Expert Systems*, Springer Verlag, (1996).

[3] It could be substituted with its definition, but retaining the name can lead to a shorter proof.

[4] Such a sequent calculus would, in general, be non terminating. However, in our framework, finite proofs are generated by the tableaux algorithm, for which termination is guaranteed.