

Explaining Subsumption by Optimal Interpolation

Stefan Schlobach

Informatics Institute
University of Amsterdam, NL,
schlobac@science.uva.nl

Abstract. We describe ongoing research to support the construction of terminologies with Description Logics. For the *explanation of subsumption* we search for particular concepts because of their syntactic and semantic properties. More precisely, the set of explanations for a subsumption $P \sqsubseteq N$ is the set of *optimal interpolants* for P and N . We provide definitions for optimal interpolation and an algorithm based on Boolean minimisation of concept-names in a tableau proof for \mathcal{ALC} -satisfiability. Finally, we describe our implementation and some experiments to assess the computational scalability of our proposal.

1 Introduction

Building ontologies is a time consuming and error-prone process and one of the bottle-necks in a number of AI applications. We have recently suggested various methods to tackle this problem. First, we investigated methods to automatically learn terminologies from data [12] and, secondly, we added explanation facilities to terminological reasoning [14]. Modern Description Logic (DL) reasoning systems efficiently answer queries, e.g., whether a concept $\exists child.\top \sqcap \forall child.Doctor$ is *subsumed* by $\exists child.(Doctor \sqcup Rich)$, i.e. whether one is a subclass of the other. Unfortunately, they do not provide explanatory information. In our approach a suitable explanation for this subsumption is that the former concept is more special than a third concept $\exists child.Doctor$ which, again, is more special than the latter. Not only is this concept, which we will call an *illustration*, strongly linked by the common vocabulary to both the subsumer and the subsumed concept, it is also the simplest such illustration.

The solutions we proposed can be generalised; given two concepts C and D we need to find an *interpolant*, i.e. a concepts I such that $\models C \sqsubseteq I \sqsubseteq D$, and where I is syntactically related to both C and D . Moreover, we need interpolants which are *optimal* with respect to particular syntactic properties such as number of concept-names or size. Algorithms for interpolation for concept subsumption in \mathcal{ALC} are well known (e.g. [11]) as \mathcal{ALC} is a notational variant of modal \mathbf{K} . In this paper we extend these algorithms in order to calculate an interpolant I with a minimal number of concept-names, i.e., where there are no interpolants built from a proper subset of the concept-names occurring in I . For this purpose we

saturate a tableau according to the usual rules for \mathcal{ALC} -satisfiability. Boolean minimisation then renders *reducts*, minimal sets of concept-names preserving the existence of interpolants. Finally, given reducts we calculate optimal interpolants.

This paper focuses on the discussion of optimal interpolants which not only lie at the core of our applications but which, as the most concise extracts of a proof for subsumption, are of more general and theoretic interest. Optimal interpolants play a key role in our ongoing efforts to support knowledge engineering, and robust algorithms are essential for the evaluation in particular applications. A prototypical implementation provides first insights to the potential of our approach, in particular our early experiments show that the methods do not easily scale up for more complex subsumption relations. To solve this problem we discuss a number of well-studied optimization techniques with respect to optimal interpolation.

The remainder of the paper is organised as follows: in Section 2 we introduce the relevant DL notation to make the paper self-contained. In Section 3 we discuss the applications of explanation in more detail. Section 4 introduces interpolation for DL, more specifically for \mathcal{ALC} , with the vocabulary \mathcal{L} of a concept, and formal notions of optimality. In Section 5 we provide an algorithm to calculate optimal \mathcal{L} -interpolants based on Boolean minimisation of concept-names w.r.t. the closure of a tableau for \mathcal{ALC} -satisfiability. We finish with an evaluation of the proposed methods in Section 6 and a brief discussion of related and further work.

2 Description Logics

We shall not give a formal introduction to Description Logics (DL) here, but point to the first two chapters of the DL handbook [1] for an excellent overview. Briefly, DLs are set description languages with concepts (usually we use capital letters), interpreted as subsets of a domain, and roles which are binary relations, denoted by small letters. \mathcal{ALC} is a simple yet relatively expressive DL with conjunction $C \sqcap D$, disjunction $C \sqcup D$, negation $\neg C$ and universal $\forall r.C$ and existential quantification $\exists r.C$. In a terminology \mathcal{T} (called TBox) the interpretations of concepts can be restricted to the *models* of \mathcal{T} by *axioms* of the form $C \sqsubseteq D$ or $C \doteq D$. Based on this model-theoretic semantics, concepts can be checked for *unsatisfiability*: whether they are necessarily interpreted as the empty set.

A TBox \mathcal{T} is called *coherent* if no unsatisfiable concept-name occurs in \mathcal{T} . Other checks include *subsumption* of two concepts C and D (a subset relation of $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ w.r.t. all models \mathcal{I} of \mathcal{T}). Subsumption between concepts C and D w.r.t. a TBox \mathcal{T} will be denoted by $\mathcal{T} \models C \sqsubseteq D$. A TBox is *unfoldable* if the left-hand side of the axioms are atomic, and if the right-hand sides (the definitions) contain no reference to the defined concept [9]. Unfolding then comes down to replacing atomic sub-concepts by their definitions. Subsumption without reference to a TBox will be denoted by *concept subsumption* and we will write $C \sqsubseteq D$.

3 Explaining Subsumption by Illustration

Recently, the DL community has shown growing interest in explanation of reasoning (e.g. [6]) to provide additional information to increase the acceptance of logical reasoning, and to give additional insights to the structure of represented knowledge. Let us consider a variant of an example introduced in [4] for the author’s “explanation as proof-fragment” strategy, where a concept $C_{ex} := \exists child. \exists child. Rich \sqcap \forall child. \neg (\exists child. \neg Doctor) \sqcup (\exists child. Lawyer)$ is subsumed by $D_{ex} := \exists child. \forall child. (Rich \sqcup Doctor)$. Instead of providing a concise and simplified extract of a formal proof as an explanation as done in [4] we suggest an alternative, more static approach, which we call *explaining by illustration*. Imagine the above information is given in natural language: *Suppose somebody has a rich grand-child, and each child has neither a child which is not a doctor nor a child which is a Lawyer. Then, this person must have a child, every child of which is either rich or a doctor.* A natural language explanation for this statement is: *The person described above must have a child every child of which is a doctor.* This intermediate statement can be considered an *illustration* of $\models C_{ex} \sqsubseteq D_{ex}$. It can be formalised as $I_{ex} := \exists child. \forall child. Doctor$, and it subsumes C_{ex} and is subsumed by D_{ex} . Moreover, it is constructed from vocabulary both in C_{ex} and D_{ex} , e.g., the information that the person’s grandchildren might be a Lawyer is irrelevant as, just from D_{ex} , we don’t know anything about grandchildren being Lawyers or not. Finally, the illustration should use a minimal number of concept-names and be of minimal size, as this increases the likelihood of it being understandable.

To explain more complex subsumption relations a single illustration might not be sufficient, and iterative explanation is needed. Here, subsumption between concepts and their illustrations could, again, be explained by new illustrations or traditional explanation of proofs which are now, though, simpler to understand.

4 Optimal Interpolation

Calculating illustrations and optimal TBox axioms are interpolation problems, more precisely, problems of finding *optimal interpolants*. Remember that an axiom $D \sqsubseteq L_D$ is correct if $\models \bigsqcup_{P \in \mathbf{P}} P \sqsubseteq L_D \sqsubseteq \prod_{N \in \mathbf{N}} \neg N$ for examples \mathbf{P} and counterexamples \mathbf{N} of a concept D , and that an illustration for the subsumption $\models C \sqsubseteq D$ was a concept I s.t. $\models C \sqsubseteq I \sqsubseteq D$. In both cases we have argued that common vocabulary and syntactic minimality is desirable.

An *interpolant* for concepts P (for the positive) and N (the negative examples), where $\models P \sqsubseteq N$, is a concept I which is more general than P but more special than N . Furthermore, I has to be built from the vocabulary occurring both in P and N . In the standard definition of interpolation [5] the vocabulary of a formula ϕ is defined as the set of non-logical symbols in ϕ . Because we use interpolants for learning and explanation we propose a stronger notion of vocabulary for concepts: including information about the context in which the non-logical symbols (such as concept- and role-names) occur. Formally, $\mathcal{L}(C)$ is a set of pairs $(A, +)^S$ or $(B, -)^S$ of concept-names A, B and polarity $+$ or $-$, labelled with sequences S of role-names. A concept-name A has positive (negative)

polarity if it is embedded in an even (odd) number of negations. \perp and \top always occur without polarity (represented by pairs $(\perp, -)$ and $(\top, -)$). $\bar{\mathcal{L}}(C)$ denotes the set $\mathcal{L}(C)$ where the polarity of each pair is interchanged (i.e. $+$ replaced by $-$ and vice versa). Furthermore, for a set S , let S^r denote that the sequence of role-names for each element of S has been extended by r . The vocabulary $\mathcal{L}(C)$ of a concept C is then defined as follows.

Definition 1. \mathcal{L} is a mapping from \mathcal{ALC} to triples of concept-names, polarities and sequences of role-names defined as follows:

- $\mathcal{L}(\top) = \mathcal{L}(\perp) := \{(\top, -)^\epsilon, (\perp, -)^\epsilon\}$
- $\mathcal{L}(C \sqcap D) := \mathcal{L}(C) \cup \mathcal{L}(D)$
- $\mathcal{L}(A) := (A, +)^\epsilon \cup \mathcal{L}(\top)$ if A is atomic
- $\mathcal{L}(C \sqcup D) := \mathcal{L}(C) \cup \mathcal{L}(D)$
- $\mathcal{L}(\exists r.C) = \mathcal{L}(\forall r.C) := \mathcal{L}(C)^r \cup \mathcal{L}(\top)^\epsilon$
- $\mathcal{L}(\neg C) := \bar{\mathcal{L}}(C)$

Take a concept $\forall r.(D \sqcap \neg \exists s.C)$. The related language is the set $\{(C, -)^{rs}, (D, +)^r, (\top, -)^\epsilon, (\top, -)^r, (\top, -)^{rs}, (\perp, -)^\epsilon, (\perp, -)^r, (\perp, -)^{rs}\}$. Note that this definition implies that $\mathcal{L}(\perp) \in \mathcal{L}(C)$ and $\mathcal{L}(\top) \in \mathcal{L}(C)$ for every concept C .

The set of interpolants w.r.t. \mathcal{L} will be denoted by $I(P, N)$. In applications where interpolants are used for explanation or learning, additional restrictions are important to identify optimal illustrations or learning targets. There are several types of syntactic restrictions, e.g., interpolants with a minimal set of concept- or role-names or of minimal size, but we will focus on concept-name optimal interpolants. To simplify the presentation we only consider concept interpolation, i.e. interpolation for concept subsumption w.r.t. empty TBoxes.

Definition 2. Let P and N be concepts, and let $\mathcal{N}(C)$ denote the set of concept-names occurring in an arbitrary concept C . A concept I is an *optimal interpolant* for P and N if $\models P \sqsubseteq I$ and $\models I \sqsubseteq N$, $\mathcal{L}(I) \subseteq \mathcal{L}(P) \cap \mathcal{L}(N)$, and if there is no interpolant I' for P and N with $\mathcal{N}(I') \subset \mathcal{N}(I)$.

Take $I_{ex} := \exists child. \forall child. Doctor$ from Section 3 which is an interpolant for $C_{ex} := \exists child. \exists child. Rich \sqcap \forall child. \neg((\exists child. \neg Doctor) \sqcup (\exists child. Lawyer))$ and $D_{ex} := \exists child. \forall child. (Rich \sqcup Doctor)$, as $\mathcal{L}(I_{ex}) = \{(Doctor, +)^{child \ child}, \dots\}$ is a subset of $\mathcal{L}(C_{ex}) \cap \mathcal{L}(D_{ex})$. Moreover, the set of concept-names in I_{ex} is $\{Doctor\}$ which is minimal, and I_{ex} is an optimal interpolant for C_{ex} and D_{ex} . Such a minimal set of concept-names will be called a *reduct*.

To define reducts we need some more notation. Let S be an arbitrary subset of the common language of two concepts P and N . The set of interpolants for P and N built from concept-names in S only will be denoted as $I_S(P, N)$. For C_{ex} and D_{ex} and a set $S = \{Doctor, Rich\}$ the set $I_{\{Doctor, Rich\}}(C_{ex}, D_{ex})$ then contains, e.g., $\exists child. \forall child. Doctor$ and $\exists child. \forall child. Doctor \sqcup Rich$. Reducts now determine smallest sets S such that $I_S(P, N) \neq \emptyset$.

Definition 3. A *reduct* for two concepts P and N is a minimal set of concept-names R to preserve existence of an interpolant, i.e. where $I_R(P, N) \neq \emptyset$ and $I_{R'}(P, N) = \emptyset$ for every $R' \subset R$.

The set $\{Doctor, Rich\}$ is not minimal, as $I_{\{Doctor\}}(C_{ex}, D_{ex}) \neq \emptyset$. On the other hand, $\{Doctor\}$ is a reduct. Also, each interpolant in $I_{\{Doctor\}}(C_{ex}, D_{ex})$ is optimal for C_{ex} and D_{ex} . This observation can be generalised:

Lemma 1. *Let R be a reduct for two concepts P and N . Every interpolant $I \in I_R(P, N)$ is optimal for P and N .*

This lemma allows to calculate reducts and interpolants separately, and is used in the algorithms in the next section.

5 Algorithms for Optimal Interpolation

Optimal interpolants will be constructed using Boolean minimisation of the concept-names needed to close an \mathcal{ALC} -tableau. This calculation can be split into three steps. First, we saturate a labelled tableau as described in Section 5.1. Secondly, we calculate reducts from tableau proofs using the algorithm of Fig. 2. Finally, optimal interpolants can be constructed from the tableau proofs and the reducts according to Fig. 3. Step 1 and 3 closely follow well-known procedures that can be found, e.g., in [2] (for the saturation of the tableau) and [8] (for the calculation of interpolants). New is the calculation of reducts in Section 5.2 and their application in Section 5.3 to ensure optimality of the calculated interpolants.

5.1 Saturating Labelled Tableaux

Concept subsumption $\models P \sqsubseteq N$ can be decided by a proof deriving a closed tableau starting from a tableau with one branch $\{(i : P)^p, (i : \neg N)^n\}$ (for an arbitrary individual i). The information whether a formula has its origin in P or N is needed to construct interpolants from the proof. Each formula stemming from P will be labelled with $(\cdot)^p$, each created from N with $(\cdot)^n$. A *formula* has the form $(i : C)^x$ where i is an individual, C a concept and $x \in \{p, n\}$ a label. The labelling mechanism follows [8]. A *branch* is a set of formulas and a *tableau* a set of branches. A formula can occur with different labels on the same branch. A branch is *closed* if it contains a clash, i.e. if there are formulas with contradictory atoms on the same individual. The notions of open/closed branches and tableaux are defined as usual and do not depend on the labels. Formulas are always assumed to be in *negation normal form*.

To calculate reducts and optimal interpolants for two concepts P and N we construct a proof from a tableau containing a branch $\{(i : P)^p, (i : \neg N)^n\}$ (for a new individual i) by applying the rules in Fig. 1 as long as possible. The rules are \mathcal{ALC} -tableau rules (adapted from those of [2]) and have to be read in the following way; suppose that there is a tableau $T = \{B, B_1, \dots, B_n\}$ with $n + 1$ branches. Application of one of the rules on B yields the tableau $T' := \{B', B_1, \dots, B_n\}$ for the (\sqcap) and (\exists) rule or $T'' := \{B', B'', B_1, \dots, B_n\}$ in case the (\sqcup) -rule has been applied. Application of one of the rules is called *to expand* a tableau or a branch. If no more rule can be applied, branches and tableaux are *saturated*. Finally, a *proof* is a sequence of tableaux T_1, \dots, T_n where each T_{i+1} has been created by application of one of the rules in Fig. 1 on a branches $B \in T_i$ (for $i, i + 1 \in \{1, \dots, n\}$), and where T_n is saturated.

(\sqcap) : if $(i : C \sqcap D)^x \in B$, but not both $(i : C)^x \in B$ and $(i : D)^x \in B$ then $B' := B \cup \{(i : C)^x, (i : D)^x\}$.
(\sqcup) : if $(i : C \sqcup D)^x \in B$, but neither $(i : C)^x \in B$ nor $(i : D)^x \in B$. then $B' := B \cup \{(i : C)^x\}$ and $B'' := B \cup \{(i : D)^x\}$.
(\exists) : if $(i : \exists r.C)^x \in B$, all other rules have been applied, and $\{(i : \forall r.C_1)^{x_1}, \dots, (i : \forall r.C_n)^{x_n}\}$ are all universal formulas for i and r in B , then $B' := B \cup \{(j : C)^x, (j : C_1)^{x_1}, \dots, (j : C_n)^{x_n}\}$, where j is new in B .

Fig. 1. Tableau rules for saturating a labelled \mathcal{ALC} -tableau (similar to [2])

5.2 Calculating Reducts

From a proof starting with $\{(i : P)^p, (i : \neg N)^n\}$ we find reducts by calculating a *maximal reduct-function*. To define such an interpolation-preserving propositional formula we need to introduce interpolants for branches and individuals. Let B be a branch, i an individual and $\{(i : C_1)^p, \dots, (i : C_k)^p, (i : D_1)^n, \dots, (i : D_l)^n\}$ the set of formulas for i in B . An *interpolant* for B and i is an interpolant for $C_1 \sqcap \dots \sqcap C_k$ and $\neg D_1 \sqcup \dots \sqcup \neg D_l$.

The set of interpolants for B and i will be denoted by $I(i, B)$, or $I_S(i, B)$ for the interpolants built from concept-names occurring in a particular set S only. Reduct-functions are then interpolation-preserving propositional formulas.

Definition 4. Let B be a branch in a proof, i an individual and ϕ a propositional formula built from conjunction, disjunction and propositional variables. Let, furthermore, $tr(v)$ denote the (unique) set of propositional variables true in a valuation v , called the *truth-set* of v . If $I(i, B) \neq \emptyset$, ϕ is a *reduct-function* for B and i if, and only if, $I_{tr(v)}(i, B) \neq \emptyset$ for any valuation $v(\phi) = T$. Otherwise, i.e., if $I(i, B) = \emptyset$, \perp is the only reduct-function.

The idea to calculate reducts is as follows: As reduct-functions determine the sets of concept-names preserving interpolation a smallest set of this kind is a reduct. If a reduct-function is maximal, i.e. implied by all reduct-function, its prime implicants determine precisely these most general, i.e. smallest sets of concept-names.¹ A maximal reduct-function is calculated from a tableau proof: all branches of the saturated tableau must close even with a reduced set of concept-names available for closure. Therefore at least one clash per branch needs to be retained and a maximal reduct-function is the disjunction of the concept-names in all clashes. For each branch in a tableau proof complex maximal reduct-functions are then constructed recursively according to Fig. 2.

Theorem 1. Let P and N be concepts and i an arbitrary individual-name. The prime implicants of $rf(i, \{(i : P)^p, (i : \neg N)^n\})$ as calculated by the algorithm described in Fig. 2 are the reducts for P and N .

¹ A prime implicant of ϕ is the smallest conjunction of literals implying ϕ (see, e.g., [10]). The term *prime implicant* refers both to the conjunction and to the set of conjuncts.

if rule = (\sqcap) has been applied on $(i : C \sqcap D)^{label}$ and B' is the new branch
 $rf(i, B) := rf(i, B')$;
if rule = (\sqcup) has been applied on $(i : C \sqcup D)^{label}$ and B' and B'' are new
 $rf(i, B) := rf(i, B') \wedge rf(i, B'')$;
if rule = (\exists) has been applied on $(i : \exists r.C)^{label}$, B' and j are new
 $rf(i, B) := rf(i, B') \vee rf(j, B')$;
if no more rule can be applied (for arbitrary x and y)
 $rf(i, B) := \top$ if there are formulas $(i : A)^x \in B, (i : \neg A)^y \in B$ s.t. $x = y$;
 $rf(i, B) := \bigvee_{(i : A)^x \in B, (i : \neg A)^y \in B} A$ otherwise; i.e. if $(x \neq y)$.

Fig. 2. $rf(i, B)$: A maximal reduct-function for a branch B and individual i

Proof. The proof consists of three parts. First, we show that $rf(i, B)$ is a reduct-function for every branch B . The proof is by induction over the tableaux in a proof, where we construct interpolants for B and i from the truth-sets of the valuation making $rf(i, B)$ true. The rules for construction of interpolants correspond to those we will later give explicitly in Fig. 3. If B is saturated there are several cases: first, if there are contradicting atoms with positive or with negative labels only, the interpolant is \perp or \top respectively, and the reduct-function is \top . If there are only clashes on atoms which occur both positively and negatively labelled in B any of the literals occurring positively is an interpolant, and the maximal reduct function is the disjunction of all the corresponding concept-names. Finally, a branch without clashes on the individual i does not have an interpolant for B and i , and \perp is the only reduct-function. If the branch B is not saturated, one of the rules of Fig. 2 must have been applied, and we can construct an interpolant for B and i from the interpolants of the newly created branches. If a disjunctive rule had been applied, two new branches B' and B'' have been created, and it can easily be checked that the disjunction of an arbitrary interpolant for B' and i with an arbitrary interpolant for B'' and i is an interpolant for B and i . The conjunctive case is even more simple, as any interpolant for the new branch B' and i is also an interpolant for B and i . The only slightly more complicated case is when an existential rule has been applied on a formula in B because we need to take into account that the interpolant for the new branch and the new individual might be \perp . In this case, however, we can show that \perp is also an interpolant for B and i , which finishes the proof.

Next, we show that $rf(i, B)$ is maximal, again by induction over the tableaux in the proof. For each branch B we show that $\phi \rightarrow rf(i, B)$ for each reduct-function ϕ of i and B . Again, if B is saturated it is easy to show that $rf(i, B)$ is maximal for each of the cases mentioned above. For a non-saturated B we again have branches B' (and possibly B''), and we can easily show that $\phi \rightarrow rf(i, B)$ whenever $\phi \rightarrow rf(i, B')$ (and possibly $\phi \rightarrow rf(i, B'')$).

Finally, we prove that the prime implicants of maximal reduct-functions for the branch $B = \{(i : P)^p, (i : \neg N)^n\}$ are the reducts for P and N . Here, we show that there is an interpolant for P and N , and that there is no interpolant for any subset of the reduct. The other direction, i.e. the fact that every reduct R is the

```

if rule =  $(\sqcap)$  has been applied on  $(i : C \sqcap D)^{label}$  and  $B'$  is the new branch
  return  $oi(i, B', R)$ ;
if rule =  $(\sqcup)$  has been applied on  $(i : C \sqcup D)^{label}$  and  $B'$  and  $B''$  are new
  if label =  $p$  return  $oi(i, B', R) \sqcup oi(i, B'', R)$ ;
  else if label =  $n$  return  $oi(i, B', R) \sqcap oi(i, B'', R)$ ;
if rule =  $(\exists)$  has been applied on  $(i : \exists r.C)^{label}$ ,  $B'$  and  $j$  are new
  if  $oi(i, B', R)$  exists
    if  $oi(j, B', R)$  exists
      if label =  $p$  if  $oi(j, B', R) = \perp$  return  $oi(i, B', R)$ ;
        else return  $oi(i, B', R) \sqcup \exists r.oi(j, B', R)$ ;
      else if label =  $n$  if  $oi(j, B', R) = \top$  return  $\top$ 
        else return  $oi(i, B', R) \sqcup \forall r.oi(j, B', R)$ ;
    else return  $oi(i, B', R)$ ;
  else if  $oi(j, B', R)$  exists
    if label =  $p$  if  $oi(j, B', R) = \perp$  return  $\perp$ ; else return  $\exists r.oi(j, B', R)$ ;
    else if label =  $n$  if  $oi(j, B', R) = \top$  return  $\top$ ; else return  $\forall r.oi(j, B', R)$ ;
  else return undefined;
if no more rule can be applied.
  if there is a clash on a concept-name  $A \in R$ 
    if there are formulas  $(i : A)^n \in B$  and  $(i : \neg A)^n \in B$  return:  $\top$ 
    else if there are formulas  $(i : A)^p \in B$  and  $(i : \neg A)^p \in B$  return:  $\perp$ 
    else return:  $\bigsqcup_{(i:A)^p \in B, (i:\neg A)^n \in B, A \in R} A \sqcup \bigsqcup_{(i:A)^n \in B, (i:\neg A)^p \in B, A \in R} \neg A$ 
  else return: undefined.

```

Fig. 3. $oi(i, B, R)$: Optimal interpolants

prime-implicant of the maximal reduct-function of B , follows immediately from the minimality of the reducts. See [13] for the technical details of the proof.

5.3 Calculating Optimal Interpolants

Given a tableau proof and a reduct R we construct optimal interpolants for each branch B and each individual i recursively according to the rules in Fig. 3. It is well-known how to calculate interpolants from a tableau proof [8,11]. If a rule had been applied on a formula $(i : C)^x$ in B for an arbitrary label x , and one or two new branches B' (and B'') have been created, the interpolant for B and i can be constructed from interpolants for B' (and B''). It can easily be checked that $oi(i, \{(i : P)^p, (i : \neg N)^n\}, R)$ is an interpolant for P and N . By Lemma 1 we now immediately know that if R is a concept-reduct for P and N , this interpolant is also optimal.

Theorem 2. Let R be a reduct for two concepts P and N . The concept $oi(i, \{(i : P)^p, (i : \neg N)^n\}, R)$ as calculated by the algorithm defined in Fig. 3 is an optimal interpolant for P and N .

Proof. Lemma 1 states that the interpolants built from concept-names in reducts are the optimal interpolants. Therefore, it suffices to show that the rules in

Fig. 3 produce interpolants. This proof follows Kracht’s proof in [8], we simply construct interpolants recursively for each branch in the proof. Full details can be found in [13].

Optimal interpolants are not unique, and a certain leeway exists for choosing suitable interpolants according to a given application. The algorithm in Fig. 3 was developed with an application to learning of terminologies in mind [12] and, for this reason, calculates very general interpolants. Whenever we have a choice we construct an interpolant in the most general way. This is the case when applying a (\exists) -rule as well as when calculating an interpolant for a saturated branch. Note that this choice might lead to rather complex concepts with a relatively big size. Our decision was to separate the two problems of optimal interpolation and rewriting of concepts with minimal size for conceptual clarity. If optimal interpolants are applied to explain a subsumption relation things might be different: first, the size of an explanation should be as small as possible and, secondly, we might also want to have different levels of generality, such as the most specific optimal interpolant. It is straightforward to adapt the algorithm of Fig. 3 to calculate smaller or more specific optimal interpolants and we plan to evaluate different strategies for both described applications in future research.

5.4 Complexity

Calculating interpolants for two \mathcal{ALC} concepts P and N is in PSPACE as we can apply the algorithm described in Fig. 3 in a depth-first way on one branch (of maximally polynomial space) at a time. This gives a simple bottom-up procedure to calculate optimal interpolants in PSPACE: for all subsets of the concept-names occurring in P and N we check whether there are interpolants or not, starting with the smallest, systematically increasing the size. Each of these checks can be done using only polynomial space. As soon as we find the first interpolant, i.e. as soon as $oi(i, B, R)$ is defined for some subset R , we know that R is a reduct, and the interpolant must be optimal. For a lower bound consider the subsumption relation $C \sqsubseteq \perp$ which has an interpolant if, and only if, C is unsatisfiable. This means that interpolation must be at least as hard as concept satisfiability in \mathcal{ALC} , which is well known to be PSPACE.

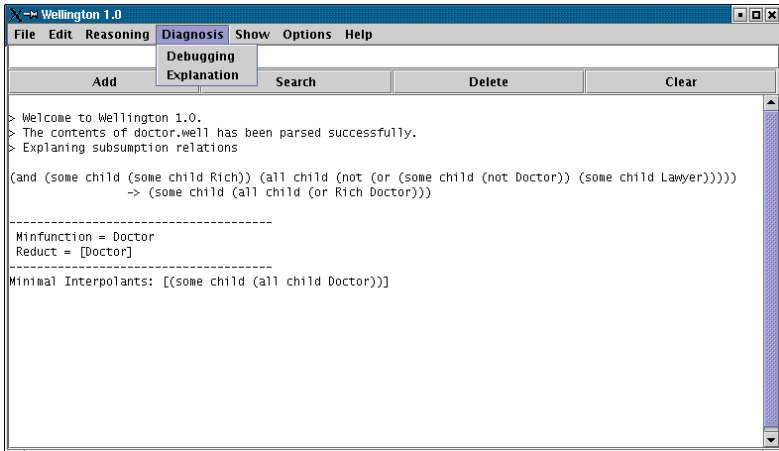
Although the problem of calculating optimal interpolants is in PSPACE the algorithm described above might be infeasible in practice. To be sure that we have calculated all optimal interpolants we might have to check all elements of the power-set of the set of concept-names, i.e., we might have to saturate an exponential number of tableaux. Instead, our approach expands a single tableau once, from which we calculate the reducts and read off the optimal interpolants. Computing reducts is the computational bottle-neck of our algorithm as we calculate prime implicants on formulas which can be exponential in the size of the concepts. Given that calculating prime implicants is NP-hard, we must ensure that the size of the reduct-function is as small as possible. Our current implementation comprises simple on-the-fly elimination of redundancies, but more evolved methods need to be investigated. Our algorithms have an exponential worst case

complexity in the size of the concepts P and N . As the number of variables in the reduct-function is linear in the number of concept-names in P and N we can calculate the prime implicants in exponential time branch by branch (instead of constructing the full reduct-function first). This simple method requires exponential space as we have to keep maximally $e^{\frac{n}{e}}$ prime implicants (of size smaller than n) in memory, where n is the number of concept-names in P and N and e the base of natural logarithm.

6 Evaluation

Explanation of subsumption by interpolation has to be evaluated with respect to two different problems: first, we have to find out whether the explanation is indeed human-understandable, and, secondly, we have to study whether the approach scales up to more than toy examples. In this paper we will concentrate on the latter, as there is relatively little testing data available. Before investing too much time in studies with human assessors of explanations we decided to focus on the computational properties of explanation by interpolation.

Implementation. We implemented optimal interpolation in Java as part of our Wellington [7] reasoning system. The program takes as input two \mathcal{ALC} concepts C and D in KRSS representation. First we use RACER to check for subsumption between C and D . If C subsumes D (or vice versa) we fully expand a tableau as described in the previous section, calculate the minimization function and the reducts, and finally read the optimal interpolants from the closed branches.



The Experiments. There is, to the best of our knowledge, no collection of subsuming concepts available to test our algorithms. To evaluate the run-times of our algorithm we therefore transformed formulas from a test-set for modal logic theorem provers provided at [3] into suitable \mathcal{ALC} subsumption relations. The

test-set contains 131 unsatisfiable concepts for several modal logics. For our experiments we chose 72 concepts which were unsatisfiable in modal logic K. These formulas are mostly of the form $\neg(\phi \rightarrow \psi)$ where ϕ and ψ are complex formulas (often also containing implications). To create a test-set for \mathcal{ALC} subsumption we simply picked an implication and translated $\neg(\phi \rightarrow \psi)$ to $\phi^t \sqsubseteq \psi^t$, where $(\cdot)^t$ is the standard translation from K to \mathcal{ALC} . The transformation is very simplistic, as there is usually a number of implications in the modal formula, and we simply picked one at random.

This test-set is not useful to evaluate the explanatory quality of the interpolants because the formulas have been created to be most compact representations for computationally difficult problems in modal reasoning. In this case, explanation by interpolation is little helpful as its main purpose is to reduce syntactic overhead. But the test-set can help to get a better understanding of the computational properties of the method.

Results. Even though the test set is considered to be trivial for current specialized DL-reasoners it creates difficulties for our naive algorithm for optimal interpolation. More concretely, from the 72 subsumption relations we fail to find optimal interpolants for more than a dozen. The reason for this is obvious: remember that we had to expand the full tableau to calculate the minimization function. This means that we always expand exponentially many branches in the number of disjunctions of implications, which does not scale up even for relatively trivial concepts.

Interpreting the Results. Our method faces the problems that modal and description logic reasoning had to solve 20 years ago, namely that naive implementation of tableau calculi without optimizations cannot deal with the exponential complexity of the reasoning problems. In our case our implementation already fails to calculate the minimization function as too many branches have to be visited. There are two approaches to deal with this problem: to ignore it, assuming that those subsumption relations that can be explained by interpolation must be simple enough to allow for our reasoning algorithms to efficiently deal with. The second alternative is to look at the optimizations that have made such a difference in the standard reasoning in Description Logics.

Optimizations. The most simple optimization would be to allow non-atomic contradictions, i.e., closure of branches on complex formulas. In this case it is unclear how to calculate the minimization function as we do not know which concept-name was responsible for the logical contradiction. Probably the most obvious optimization is to not fully expand the tableau but to stop expanding once a contradiction has been found. Unfortunately, we lose completeness in both cases. This means that we do not calculate reducts, and therefore optimal interpolants, any more. A simple example is $(B \sqcap A) \sqcup A \sqsubseteq A \sqcup B$. If branch 1 is closed with B we calculate a reduct $A \wedge B$ and an interpolant $A \sqcup B$, which is not minimal as A is already an interpolant.

As a result, we would claim that our proposed way of calculating optimal interpolants is inherently intractable. If this is considered a problem, i.e., if we want to explain more complex subsumption relations we will have to accept the fact that we can approximate reducts, but not find all reducts for certain. In that case reasoning becomes immediately much more efficient, e.g., we could easily solve all of the 72 problems of our test-suite when we stopped expanding the branches as soon as we found a contradiction.

Let us discuss some of the more sophisticated optimization techniques as they are discussed, for example, in Chapter 9 of [1]. Due to lack of space we cannot introduce the techniques in more detail. One of the most important optimizations is to *pre-process* the concepts prior to running the tableau engine. For explanation this would not work because preprocessing, such as normalization, destroys the structure of the subsumption relation. This, however, is what makes the optimal interpolant an explanation, namely that it is a simplified version of the original concepts (both in terms of vocabulary and of structure).

Caching, on the other hand, is a technique that can, and has been applied for optimal interpolation, as we can cache information about satisfiability, but also about the minimization function and the interpolant for a branch. We implemented *semantic branching* in our “sub-optimal” interpolation of interpolation with great computational gain. However, we will have to study more carefully whether minimization function could still be calculated from the tableau, and whether the interpolant still has a useful structure for explanation.

To integrate *local simplification* into our calculus we would need to explicitly give rules to calculate the minimization function for each simplification. Assume a simplification has been applied on two concepts $(A \sqcup B) \sqcap (A \sqcup C)$ and $\neg A$ in a branch, with concepts B and C added to create a new branch B' . In this case we would have to add A conjunctively to the minimization function for B' . We assume that *backtracking* could be implemented because the interpolant for a redundant branch will probably be redundant, but we do not have a formal proof for this claim.

What remains as a further interesting optimization is the use of *heuristics*. In our case, we could use statistical information to strike the balance of calculating a good minimization function and having an efficient algorithm. We could, for example, expand branches as long as we find a contradiction on a concept-name which has already been frequently responsible for the closure of other branches.

7 Conclusions

We introduce an algorithm to find interpolants with a minimal number of concept-names for two concepts in the description logic \mathcal{ALC} , with a rigid definition of the common vocabulary. The principal novelty is that we minimise the number of concept-names in order to find most simple interpolants. These optimal interpolants are used in applications as diverse as explanation and learning of terminologies.

We presented a prototypical implementation of the algorithms, in order to evaluate explanation by interpolation. In the absence of real-life data we focused on the assessment of the computational properties of the described methods, and we discussed a number of optimizations to deal with the inherent non-tractability.

For future research we plan to extend the algorithms to more expressive languages, and to apply optimal interpolation in the automatic construction of ontologies from assertions as described in [12]. An open problem is how to calculate interpolants of minimal size, as redundancy elimination will be crucial for optimal interpolants to be useful in newly learned terminologies or as illustrations.

Acknowledgment. This research was supported by the Netherlands Organization for Scientific Research (NWO) under project number 220-80-001. Thanks to the anonymous referees for some valuable comments.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
3. B. Beckert and R. Goré. Modleantap and modleantest.pl, 1998. <http://i12www.ira.uka.de/modlean>.
4. A. Borgida, E. Franconi, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Explaining *ACC* subsumption. In *DL-99*, pages 37–40, 1999.
5. W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22:269–285, 1957.
6. Minutes of the DL Implementation Group Workshop. <http://dl.kr.org/dig/minutes-012002.html>, visited on January 9, 2003.
7. U. Endriss. Reasoning in description logics with WELLINGTON 1. 0. In *Proceedings of the Automated Reasoning Workshop 2000*, London, UK, 2000.
8. M. Kracht. *Tools and Techniques in Modal Logic*. North Holland, 1999.
9. B. Nebel. Terminological reasoning is inherently intractable. *AI*, 43:235–249, 1990.
10. W.V. Quine. The problem of simplifying truth functions. *American Math. Monthly*, 59:521–531, 1952.
11. W. Rautenberg. Modal tableau calculi and interpolation. *Journal of Philosophical Logic*, 12:403–423, 1983.
12. S. Schlobach. *Knowledge Acquisition in Hybrid Knowledge Representation Systems*. PhD thesis, University of London, 2002.
13. S. Schlobach. Optimal interpolation. Technical Report PP-2003-23, Universiteit van Amsterdam, ILLC, 2003. Beta Preprint Publication.
14. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.