ABox abduction in $\mathcal{ALC}$ using a DL tableau

Ken Halland
School of Computing
University of South Africa
Pretoria, South Africa
hallakj@unisa.ac.za

Katarina Britz
Centre for Artificial Intelligence Research
UKZN and CSIR Meraka Institute
South Africa
arina.britz@meraka.org.za

ABSTRACT
The formal definition of abduction asks what needs to be added to a knowledge base to enable an observation to be entailed by the knowledge base. ABox abduction in description logics (DLs) asks what ABox statements need to be added to a DL knowledge base to allow an observation (also in the form of ABox statements) to be entailed. Klarman *et al* have provided an algorithm for performing ABox abduction in the description logic $\mathcal{ALC}$ by converting the knowledge base and observation to first-order logic, using a connection tableau to obtain abductive solutions, and then converting these back to DL syntax. In this paper we describe how this can be done directly using a DL tableau.

Categories and Subject Descriptors
1.2.8 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms
Algorithms, Theory

Keywords
Description logics, abduction, semantic tableaux

1. INTRODUCTION
Abduction is a form of non-standard reasoning where explanations are sought for certain observations in the context of some background knowledge. This is as opposed to deduction – the standard form of reasoning where the logical consequences of some knowledge are determined.

A typical use of abduction is in the process of medical diagnosis. Say a patient displays some symptoms. A doctor uses abductive reasoning to generate hypotheses about the possible ailment(s) causing the symptoms. These hypotheses can then be tested by collecting corroborating evidence so that deduction can be used to make a correct diagnosis.

Abductive reasoning is *non-monotonic* in that the solutions to an abduction problem involving some background knowledge and an observation may no longer hold if we add new statements to the background knowledge. For example, if a doctor observes further symptoms, this may change her hypotheses of the ailment causing them. Deductive reasoning on the other hand is *monotonic* in that conclusions are not invalidated by new information.

Different forms of abduction have been formally defined in different logics. In their programmatic paper, Elsenbroich *et al* [6] define and describe various forms of abduction in description logics (DLs). A number of researchers have taken up the challenge and developed algorithms for some of these forms of abduction in selected DLs [4, 5].

In particular, Klarman *et al* [9] have provided a resolution-based algorithm and a tableau-based algorithm for performing ABox abduction in $\mathcal{ALC}$. The tableau-based algorithm involves translating the knowledge base and the observation for an abductive problem from its DL specification to first-order logic (FOL), then constructing a FOL connection tableau and harvesting abductive solutions from open branches. These solutions are then translated back to a DL notation.

In this paper, we describe an algorithm for doing this directly by means of a DL semantic tableau. A potential advantage of this approach over Klarman's [9] is that it can utilise optimisation techniques used in DL tableaux.

Section 2 describes the syntax and semantics of the descriptions logics $\mathcal{ALC}$ and $\mathcal{ALE}$, Section 3 includes a definition of ABox abduction in $\mathcal{ALC}$, and Section 4 gives a description of the standard semantic tableau algorithm for $\mathcal{ALC}$. Section 5 then describes how this algorithm can be adapted to perform ABox abduction. It includes subsections describing the complexity of the algorithm, and its soundness and (in)completeness. Finally, Section 6 discusses the prospects for future work.

This paper is an extended version of a preliminary study presented at the 2012 Description Logic workshop [7].

2. DESCRIPTION LOGICS
Description Logics (*DLs*) are a family of fragments of first-order logic, suitable as knowledge representation formalisms and amenable to the implementation of efficient reasoners

[1]. There is a trade-off between the expressivity of different DLs and the efficiency of the algorithms that have been defined to reason over them. For our purposes, we are interested in two forms of description logic, $\mathcal{ALC}$ and $\mathcal{ALE}$.

It has been shown that $\mathcal{ALC}$ is a restricted variant of the modal logic **K** and so many results of modal logic are directly transferrable to DLs, and *vice versa* [12].

2.1 $\mathcal{ALC}$

Syntax: The language of $\mathcal{ALC}$ allows three sets of names for *concepts*, *roles* and *individuals*. Apart from these names, the language includes symbols for two special concepts: $\top$ and $\bot$ (called *top* and *bottom* respectively). These are combined using the *constructors* $\neg, \sqcap, \sqcup, \exists$ and $\forall$ (called *negation*, *conjunction*, *disjunction*, *existential restriction* and *universal restriction* respectively), to form *concept descriptions* as follows:

$$C ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$$[1]

Using such concept descriptions, a *knowledge base* can be specified by a set of *statements* partitioned into an *ABox* and a *TBox*. *ABox assertions* are statements of the form $C(I)$ and $R(I, J)$ (called *concept assertions* and *role assertions*, respectively), and *TBox axioms* are statements of the form $C \sqsubseteq D$ (sometimes called *general concept inclusions*, or *GCIs*).

The example below serves as a running example. Admittedly it is not very good knowledge representation, but we have chosen the given formulation for the sake of illustration.

Example 1. The following knowledge base is intended to express the ideas that Influenza A is a form of influenza, Malaria vivax is a form of malaria (caused by *Plasmodium vivax*), and someone infected with influenza or malaria will be feverish:

$\mathsf{Influenza}(\mathsf{FLU\_A})$
$\mathsf{Malaria}(\mathsf{MAL\_V})$
$\exists \mathsf{infectedWith}.\mathsf{Influenza} \sqcup \exists \mathsf{infectedWith}.\mathsf{Malaria} \sqsubseteq \mathsf{Feverish}$

Semantics: The semantics of $\mathcal{ALC}$ is defined in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain* consisting of a (possibly infinite but non-empty) set of values and $\cdot^{\mathcal{I}}$ is a function which interprets the names in terms of the domain as follows: Each concept name A is mapped to a set of values $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role name R is mapped to a set of ordered pairs $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name I is mapped to a value $I^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Top and bottom are interpreted as $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$ in all interpretations.

Such an interpretation is extended inductively to complex concept descriptions as specified in Table 1. Axioms of the form $C \sqsubseteq D$ and assertions of the form $C(I)$ and $R(I, J)$ are *true* in an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $I^{\mathcal{I}} \in C^{\mathcal{I}}$ and $\langle I^{\mathcal{I}}, J^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$, respectively.

[1]In this and the following specifications, A represents a concept name, C and D represent arbitrary concept descriptions, and R represents an arbitrary role name. Further on, I and J are used to represent arbitrary individual names.

Concept description	Interpretation
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists R.C$	$\{x \mid \exists y : \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\forall R.C$	$\{x \mid \forall y : \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$

Table 1: Interpretation of concept descriptions

An interpretation $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K}$ if all the statements (i.e. assertions and axioms) of $\mathcal{K}$ are true in $\mathcal{I}$. A concept description C is *satisfiable* with respect to a knowledge base $\mathcal{K}$ if there is some model of $\mathcal{K}$ such that the interpretation of C is not empty. A statement ϕ is *entailed* by a knowledge base $\mathcal{K}$ if ϕ is true in all models of $\mathcal{K}$, in which case we write $\mathcal{K} \models \phi$. In an abuse of notation, we often write $\mathcal{K} \models \Phi$ where Φ is a set of statements. By this we mean that $\mathcal{K} \models \phi$ for all $\phi \in \Phi$. A knowledge base $\mathcal{K}$ is *consistent* if it admits a model.

2.2 $\mathcal{ALE}$

The description logic $\mathcal{ALE}$ is a subset of $\mathcal{ALC}$. In particular, it omits the disjunction constructor $\sqcup$, and only allows primitive negation, i.e. negation of concept names. Concept descriptions are therefore of the form:

$$C ::= \top \mid \bot \mid A \mid \neg A \mid C \sqcap D \mid \exists R.C \mid \forall R.C$$

Otherwise, the syntax and semantics of $\mathcal{ALE}$ are identical to those of $\mathcal{ALC}$.

3. ABDUCTION

An abduction problem is normally specified in terms of an observation (in the form of one or more statements) which is not entailed by some background knowledge (i.e. a set of statements), and asks what needs to be added to the background knowledge to entail the observation.

Example 2. Using the knowledge base given in Example 1, say we observe that John is feverish. An abduction problem would be to ask what should be added to the knowledge base to allow us to infer $\mathsf{Feverish}(\mathsf{JOHN})$. We expect abduction to allow us to hypothesize that John is infected with influenza, i.e. $\exists \mathsf{infectedWith}.\mathsf{Influenza}(\mathsf{JOHN})$, or that he is infected with malaria, i.e. $\exists \mathsf{infectedWith}.\mathsf{Malaria}(\mathsf{JOHN})$. More specific hypotheses would be that he is infected with Influenza A, i.e. $\mathsf{infectedWith}(\mathsf{JOHN}, \mathsf{FLU\_A})$, or that he is infected with Malaria vivax, i.e. $\mathsf{infectedWith}(\mathsf{JOHN}, \mathsf{MAL\_V})$. The reader might like to check that adding any of these assertions to the knowledge base will allow us to infer $\mathsf{Feverish}(\mathsf{JOHN})$.

One problem with a formal definition of abduction is how to narrow down the possibly infinite number of solutions to an abduction problem. Various criteria have been defined for this purpose. Like other authors [6, 9], we restrict our attention to the following three:

(i) *Consistency:* A solution should not create a contradiction with the background knowledge.

(ii) *Relevance:* A solution should be expressed in terms of the background knowledge; it shouldn't introduce an independent theory.
(iii) *Minimality:* A solution should not hypothesize more than necessary.

Example 3. The solutions given in Example 2 are consistent, relevant and minimal (i.e. minimal at least in a syntactic sense). The following solutions do not comply with these criteria:

(i) $\{\neg\mathsf{Malaria}(\mathsf{MAL\_V})\}$

If this assertion were added to the knowledge base, it would cause a contradiction, and would allow us to infer anything. But this would not be a helpful solution.

(ii) $\{\exists\mathsf{infectedWith}.\mathsf{ScarletFever} \sqsubseteq \mathsf{Feverish}, \exists\mathsf{infectedWith}.\mathsf{ScarletFever}(\mathsf{JOHN})\}$

This is an abductive solution, since if both these statements were added to the knowledge base, it would allow us to infer $\mathsf{Feverish}(\mathsf{JOHN})$. However, it would also allow us to make this inference *independently of the knowledge base* and is therefore not a relevant solution. (Incidentally, the observation itself, in this case $\{\mathsf{Feverish}(\mathsf{JOHN})\}$, is also always a non-relevant solution, since adding it to the knowledge base would allow the observation to be trivially inferred.)

(iii) $\{\exists\mathsf{infectedWith}.\mathsf{Influenza}(\mathsf{JOHN}), \exists\mathsf{infectedWith}.\mathsf{Malaria}(\mathsf{JOHN})\}$

Although this is a valid solution, it is not minimal because it hypothesizes too much, namely that John is infected with both influenza and malaria.

3.1 ABox abduction

As stated in the introduction, attempts have been made to define abduction and implement reasoners that can make abductive inferences in many logics, including description logics. ABox abduction (as opposed to general or so-called *knowledge base* abduction [6]) asks what ABox assertions need to be added to a DL knowledge base to allow an observation (also in the form of ABox assertions) to be inferred.

The astute reader will note that apart from not being relevant, Example 3(ii) is also not an ABox abduction solution, since it contains a TBox axiom.

Definition 1. Given a knowledge base $\mathcal{K}$ and a set of ABox assertions Φ (both in $\mathcal{ALC}$) such that $\mathcal{K}$ does not entail Φ and $\mathcal{K} \cup \Phi$ is consistent, then a set of ABox assertions Θ (in $\mathcal{ALE}$) is an abductive solution for $(\mathcal{K}, \Phi)$ if $\mathcal{K} \cup \Theta \models \Phi$.

(Note that we follow Klarman [9] in only allowing solutions in the less expressive $\mathcal{ALE}$. This is explained in Section 5.1.)

We can narrow down the solutions in three ways:

(i) *Consistency*: $\mathcal{K} \cup \Theta$ is consistent.
(ii) *Relevance*: Φ is not entailed by Θ.
(iii) *Minimality*: We distinguish two kinds of minimality:
  (a) *Syntactic*: No proper subset of Θ is a solution.
  (b) *Semantic*: There is no non-equivalent solution Θ' such that $\mathcal{K} \cup \Theta \models \mathcal{K} \cup \Theta'$.

Note that our definition of semantic minimality induces a partial ordering on the set of solutions, and that there can be a number of semantically minimal (non-equivalent) solutions to a particular abductive problem. We say that a solution Θ is *closer to semantic minimality* than a solution Θ' if $\mathcal{K} \cup \Theta' \models \mathcal{K} \cup \Theta$ and $\mathcal{K} \cup \Theta \not\models \mathcal{K} \cup \Theta'$.

4. SEMANTIC TABLEAU ALGORITHM

The standard semantic tableau algorithm for description logics (and for $\mathcal{ALC}$ in particular) is based on the semantic tableau algorithm for modal logics [12]. In fact, numerous improvements and optimisations (some mentioned below) have been developed for this algorithm in the context of description logics.

The basic algorithm is a decision procedure for the consistency of a knowledge base, and tries to find (i.e. construct) a model of the knowledge base by applying so-called *expansion rules* to its statements. The expansion rules only apply to assertions, so before the algorithm can commence, the TBox axioms in the knowledge base must be converted to concept assertions by a process called *internalisation*. In particular, each axiom $C \sqsubseteq D$ is written as $\neg C \sqcup D$. These are called *universal concepts* because they are applied to all the individual names used in the knowledge base and are added to the set of assertions on which the algorithm works.

If the algorithm detects a *contradiction* (also called a *clash*), i.e. the current set of assertions contains a concept assertion and its negation, it backtracks and tries another branch of its search. If it gets to a point where the current set of assertions are *saturated*, i.e. no more expansion rules can be applied and there is no contradiction, then a model has been found, the algorithm terminates and reports that the original knowledge base is consistent.

Here are some comments on Algorithm 1:

Line 1: Function negNF takes a set of ABox assertions and transforms all its concepts into negation normal form. This is an equivalent form for any concept description, where negation is "pushed inwards" so that it only appears before concept names.
Line 2: Function $\mathsf{universal}$ takes a set of TBox axioms and transforms them into universal concepts. In particular, each axiom $C \sqsubseteq D$ is written as $\neg C \sqcup D$ and then negNF is called to transform it into negation normal form. (In some implementations of the tableau algorithm, all the TBox axioms are converted into one long universal concept. We rather store them as separate concepts – one per TBox axiom – to save having to repeatedly expand the long concept.) Note that U is a global variable that is used by function $\mathsf{isConsistent}$.
Lines 3 to 5: The so-called *GCI rule* is applied to each individual mentioned in the ABox to assert that they are members of all the universal concepts.

```
   Input  : Knowledge base consisting of ABox and
            TBox
   Output: true if the knowledge base is consistent,
           false otherwise
 1 iA ← negNF(ABox);
 2 U ← universal(TBox);
 3 foreach individual I used in iA do
 4 |   foreach concept D in U do
 5 |   |   iA ← iA ∪ {D(I)}
 6 return isConsistent(iA)

 7 function isConsistent(in A): boolean
 8 while true do
 9 |   if A contains assertions of the form C(I) and
   |   ¬C(I) then
10 |   |   return false
11 |   else if A contains an assertion of the form
   |   C ⊓ D(I) then
12 |   |   A ← A ∪ {C(I), D(I)}
13 |   else if A contains an assertion of the form
   |   ∃R.C(I) then
14 |   |   choose a new individual name J not used in A;
15 |   |   A ← A ∪ {R(I, J), C(J)};
16 |   |   foreach concept D in U do
17 |   |   |   A ← A ∪ {D(J)}
18 |   else if A contains assertions of the form ∀R.C(I)
   |   and R(I, J) then
19 |   |   A ← A ∪ {C(J)}
20 |   else if A contains an assertion of the form
   |   C ⊔ D(I) then
21 |   |   return isConsistent(A ∪ {C(I)}) or
   |   |   isConsistent(A ∪ {D(I)})
22 |   else
23 |   |   return true
```

Algorithm 1: Semantic tableau algorithm for $\mathcal{ALC}$

Line 7: Function isConsistent takes a set of assertions A as an in parameter to ensure that its correct contents are used when the algorithm backtracks from a branch with a contradiction.

Lines 9 & 10: Test for a contradiction (or clash), and if so, backtrack.

Lines 11, 13, 18 & 20: The expansion rules are applied to split complex concepts into their component parts. Each rule has an additional condition (not given in the algorithm) to prevent it from being applied more than once to the same assertion. For example, for the $\sqcap$-rule in Line 11, the additional condition is: *... and* A *does not contain both* $C(I)$ *and* $D(I)$.

Lines 11 & 12: This is the $\sqcap$-rule. An assertion involving a conjunction is split into two assertions which are added to A.

Lines 13 to 17: This is the $\exists$-rule. It adds a role assertion and a concept assertion for a new (dummy) individual to the current set of assertions (on the additional condition that such assertions are not already present). The GCI rule is also applied to assert that the new individual is a member of all the universal concepts.

Lines 18 & 19: This is the $\forall$-rule. If the current set of assertions contains the universal restriction of an individual and a role assertion involving that individual as the first participant, then a concept assertion of the second participant in the role assertion is added.

Lines 20 & 21: This is the $\sqcup$-rule. It calls isConsistent recursively, once for each of the disjuncts. The recursion allows backtracking to these branching points. Note that the logical *or* between the two calls generally uses short-circuit boolean evaluation to ensure that the algorithm stops if it finds a satisfying model in the first branch.

Line 23: If no expansion rules can be applied, the set of assertions is saturated, and represents a model of the original knowledge base. Terminate the entire process and return *true* to indicate that the knowledge base is consistent.

An additional optimisation rule is generally included before the $\sqcup$-rule to deal with a simplified version of disjunction: *If* A *contains* $C \sqcup D(I)$ *as well as the complement of* $C(I)$ *(or* $D(I)$*), then* $D(I)$ *(or* $C(I)$ *respectively) is added to* A. This saves branching and having to backtrack. This rule is generally inserted between lines 10 and 11 to ensure that it is always applied before the other rules.

In $\mathcal{ALC}$ it is possible to specify a knowledge base that has infinite models (by means of a so-called *cyclic* TBox). This issue is dealt with in the semantic tableau algorithm by a technique called *blocking*, which essentially detects when more than one individual is asserted to belong to the same concepts. This test is performed whenever the set of assertions is augmented (indicated by the set union operator in Algorithm 1).

For a more detailed description of the semantic tableau algorithm for $\mathcal{ALC}$, the reader is referred to the Handbook of Knowledge Representation [2].

The algorithm described above performs *consistency checking* of a knowledge base. It can easily be adapted to perform the related reasoning task of *instance checking*, i.e. deciding whether a concept assertion is entailed by a knowledge base, as follows: The negation of the concept assertion being tested is added to the knowledge base and the algorithm described above is executed. If the resulting knowledge base is consistent, we conclude that the assertion is not entailed by the knowledge base (and *vice versa*).

5. ADAPTION OF THE TABLEAU ALGORITHM FOR ABOX ABDUCTION

The basic idea is to perform instance checking of an observation in the context of a knowledge base. If the observation does not follow from the knowledge base, the tableau will not close. The ABox assertions that would close all such open branches (if added to the original set of assertions), form the abductive solutions.

The standard semantic tableau algorithm is therefore extended to do a complete search for models, i.e. it doesn't terminate when the first open branch is attained. Every time an open branch is attained, the current set of assertions (representing a model of the original knowledge base) is stored, the algorithm backtracks and continues its search.

We immediately have a problem with the standard algorithm for instance checking if we allow multiple assertions and/or role assertions in the observation, as permitted by Definition 1, because instance checking only deals with a single concept assertion.

The problem is that we need to negate the observation. Such a set of assertions is actually the conjunction of its elements, so its negation needs to be a disjunction of the negations of its elements. Unfortunately, DL syntax does not allow us to express the disjunction of assertions, particularly when they involve different individuals.

To deal with this problem, we store the negations of the assertions of the observation in a separate set (instead of adding them to the set of assertions from which the semantic tableau algorithm starts). Whenever all other rules cannot be applied, we select one of the assertions in this set, and branch with it. This works exactly like the application of the $\sqcup$-rule.

The other problem is if the observation involves a role assertion. For example, we want to be able to deal with observations such as hasSymptom(JOHN, INTERMITTENT_FEVER). Once again, we need to add the negation of such an assertion to the knowledge base, but we cannot express the negation of a role assertion in $\mathcal{ALC}$. To get around this, we augment our notation to allow negated role assertions, and store them as $\neg R(I, J)$.

We also want to be able to produce abductive solutions that involve role assertions. For example, we want to be able to infer the more specific solutions mentioned in Example 2, namely infectedWith(JOHN, FLU_A) and infectedWith(JOHN, MAL_V). Stated more generically: Consider a knowledge base containing the assertion $\forall R.A(I)$, and say we want an abductive explanation of the observation $A(J)$. An obvious solution is $R(I, J)$. But our abductive solutions are always the complements of assertions needed to close the open branches of a semantic tableau. Since the standard tableau algorithm does not infer negated role assertions, this solution will not be generated.

Our trick is to add $R(I, J)$ to the current set of assertions whenever it is saturated and contains an assertion of the form $\forall R.C(I)$, for all non-dummy individuals J in the current set of assertions. If any of these attempts are inconsistent, then $R(I, J)$ is part of an abductive solution, because it closes the branch.

When all the models have been extracted, Reiter's minimal hitting set algorithm [11] is used to generate abductive solutions from them. Simply put, one unexpanded concept assertion[2] (involving a non-dummy individual) is chosen from each model found by the extended algorithm. Each combination of the complements of such assertions forms an abductive solution.

Finally the algorithm outputs all solutions that are consistent with the knowledge base and that are relevant.

[2]By an *unexpanded concept assertion* we mean a concept assertion to which an expansion rule has not been applied.

```
   Input  : ABox, TBox and Obs
   Output: ABox abduction solutions
 1 iA ← negNF(ABox);
 2 iO ← negNF(¬Obs);
 3 U ← universal(TBox);
 4 foreach individual I used in iA do
 5 |   foreach concept D in U do
 6 |   |   iA ← iA ∪ {D(I)}
 7 M ← {};
 8 SSet ← {};
 9 extendedST(iA, iO);
10 if M = {} then
11 |   print "The observation follows from the
   |   knowledge base";
12 |   return SSet
13 minimalHS(M, H);
14 foreach S in H do
15 |   if isConsistent(iA ∪ S) and isRelevant(S, Obs) then
16 |   |   SSet ← SSet ∪ {S}
17 return SSet

18 procedure extendedST(in A, in O)
19 while true do
20 |   if A contains assertions of the form C(I) and
   |   ¬C(I) then
21 |   |   return
22 |   else if A contains an assertion of the form
   |   C ⊓ D(I) then
23 |   |   A ← A ∪ {C(I), D(I)}
24 |   else if A contains an assertion of the form
   |   ∃R.C(I) then
25 |   |   choose a new individual name J not used in A;
26 |   |   A ← A ∪ {R(I, J), C(J)};
27 |   |   foreach concept D in U do
28 |   |   |   A ← A ∪ {D(J)}
29 |   else if A contains assertions of the form ∀R.C(I)
   |   and R(I, J) then
30 |   |   A ← A ∪ {C(J)}
31 |   else if A contains an assertion of the form
   |   C ⊔ D(I) then
32 |   |   extendedST(A ∪ {C(I)}, O);
33 |   |   extendedST(A ∪ {D(I)}, O)
34 |   else if O ≠ {} then
35 |   |   D ← next(O);
36 |   |   extendedST(A ∪ {D}, O)
37 |   else
38 |   |   if A contains an assertion of the form
   |   |   ∀R.C(I) then
39 |   |   |   foreach J in A and O do
40 |   |   |   |   if not isConsistentO(A ∪ {R(I, J)}, O)
   |   |   |   |   then
41 |   |   |   |   |   A ← A ∪ {¬R(I, J)}
42 |   |   M ← M ∪ {unexpanded(A)};
43 |   |   return
```

Algorithm 2: ABox abduction algorithm for $\mathcal{ALC}$

Here are some comments on Algorithm 2:

Line 2: Set iO is used to store the initial set of negated observations.

Lines 3 & 7: Both U and M are global variables used by procedure extendedST. U is the set of universal concepts (as in Algorithm 1) and M is a set of models (where each model is a set of unexpanded assertions obtained from an open branch).

Line 8: SSet is a set of all the solutions. It is initialised as empty.

Lines 10 to 12: If the observation is entailed by the knowledge base, then all branches will close and M will be empty. This means that we are not dealing with a proper abduction problem.

Line 13: M is sent to procedure minimalHS to generate the minimal hitting sets and store them (in fact, the negations of their assertions) in H. minimalHS ensures the syntactic minimality of solutions.

Line 15: Function isConsistent is the same as the one in Algorithm 1. isRelevant is also a version of the semantic tableau procedure that determines whether the observation is not entailed by the solution.

Lines 16 & 17: Each consistent and relevant solution is added to the set of all solutions, which are finally returned as the output of the algorithm.

Line 20: Contradictions are detected between complementary concept assertions. Instead of handling $R(I, J)$ and $\neg R(I, J)$ as a contradiction, we rather implement a variation of the optimization rule described above for Algorithm 1. (See the discussion below these comments.)

Lines 22 to 30: These are identical to the corresponding expansion rules of Algorithm 1.

Lines 31 to 33: The $\sqcup$-rule is different to that in Algorithm 1 in that it searches both branches, even if one or more models are found in the first branch.

Lines 34 to 36: If no other rules can be applied, the algorithm extracts one negated assertion from O and branches with it. Function next removes such an assertion from O and returns it.

Lines 38 to 41: These lines are to allow role assertions to be generated as possible solutions. note that, like all the other expansion rules, we need an additional condition (not shown in the algorithm) to prevent this rule from being applied when the assertion involving universal restriction has been expanded already. All possible candidates for J are tried, and the negations of any $R(I, J)$s that cause the branch to close are added to A. Function isConsistentO is like isConsistent except that it takes O as a second parameter and includes lines 34 to 36 (and the optimisation rule explained below).

Line 42: Whenever an open branch is attained, the current set of unexpanded assertions is added to M and the algorithm backtracks.

The additional optimisation rule explained in Algorithm 1 for absorbing one of a pair of complementary concepts involved in a disjunction, is again not indicated. It should be inserted as early as possible in the code, namely between lines 21 and 22. We can use a similar optimisation rule to get rid of as many assertions in O as possible: *If* A *contains an assertion* $C(I)$ *(or* $R(I, J)$*), and* O *contains the complement of* $C(I)$ *(or of* $R(I, J)$ *respectively), then remove the corresponding assertion from* O. This can avoid a lot of branching and backtracking, and should also be inserted as early as possible in the code, namely just after the abovementioned one.

Although extendedST also implements blocking (whenever assertions are added to A by means of the set union operator), this is not used in any way for the generation of solutions. The argument is as follows: Since $\mathcal{ALC}$ has the *finite model property* ([1], Chapter 5), every knowledge base that has an infinite model (handled by blocking) also has at least one finite model (represented by an open branch of the tableau). Since our algorithm closes all open branches and so removes all finite models, the infinite models will also be removed.

5.1 Solutions only in $\mathcal{ALE}$

Our algorithm suffers from the same problem described in [9], namely that the abductive solutions do not contain disjunctions, i.e. assertions of the form $C \sqcup D(I)$. For example, it does not generate the following solution to the problem described in Example 2: (∃infectedWith.Influenza) ⊔ (∃infectedWith.Malaria)(JOHN). Note that a solution with such a disjunction is closer to semantic minimality than the corresponding two solutions with the individual disjuncts.

One reason for this problem is that our algorithm only considers unexpanded concept assertions for forming solutions. Allowing expandable concept assertions to be selected for solutions would allow some disjunctions, but not all. For example, say we replaced the axiom of Example 1 with the two axioms ∃infectedWith.Influenza ⊑ Feverish and ∃infectedWith. Malaria ⊑ Feverish. In this case, the solution with the disjunction above would be a valid solution, but would not be generated.

One could construct some such solutions from their constituent parts, e.g. $C \sqcup D(I)$ could be constructed from $C(I)$ and $D(I)$, but more complex solutions involving disjunctions inside quantifiers would be more difficult, e.g. $\exists R.(C \sqcup D)(I)$ will not be generated when $\exists R.C(I)$ and $\exists R.D(I)$ are.

Nevertheless, like Klarman *et al* [9], we get around the problem by defining it away: We define ABox abduction in $\mathcal{ALC}$ as only providing solutions in $\mathcal{ALE}$ which does not allow disjunctions.

5.2 Complexity

The complexity of the standard semantic tableau algorithm for consistency checking with general TBoxes in $\mathcal{ALC}$ is ExpTime ([2], Section 3.5.2). In our extended semantic tableau (in procedure extendedST), the worst case involves maximal branching where every branch is open, since we have to store all the assertion sets of all open branches. Nevertheless, the maximum number of branches is linear in the size of the initial assertion set (as measured in [2]), and the number of assertions in each such branch is also linear in the size of the initial assertion set. This means that the space required to store all the assertion sets in all the open branches is polynomial in the size of the initial assertion set. This at least

means that the space requirements don't blow up to ExpSpace, which means that the extended semantic tableau algorithm is at worst in ExpTime.

Reiter's minimal hitting set algorithm (in general) is NP-complete [13]. In our case (in procedure minimalHS), the number of sets and their size is polynomial in the size of the initial assertion set. This means that the time required in our case is also in NP.

Finally, the algorithm invokes the functions isConsistent and isRelevant twice for each candidate solution. Although the space required for isRelevant is only polynomial (because it does not deal with the TBox), isConsistent is in ExpTime in the worst case because it must deal with the TBox. Since the number of hitting sets is polynomial in the size of the initial assertion set, the total space requirement for this process is in ExpTime.

The entire algorithm is therefore in ExpTime.

5.3 Soundness and completeness

Taking Definition 1 as the standard for ABox abduction in $\mathcal{ALC}$, Algorithm 2 is sound but not complete.

It is sound because all solutions that it generates are proper abduction solutions according to the definition. Consider the following argument: Each solution is a set comprised of the complements of assertions in the open branches of the extended semantic tableau, such that each open branch has a representative in the solution. So if the assertions of such a solution were to be added to the knowledge base (and the satisfiability test were to be performed again), all branches would close, indicating that the observation is now entailed by the knowledge base. This is precisely the definition of an abduction solution.

It is not complete due to the often infinite number of solutions to an abduction problem. Narrowing down the spectrum of solutions by means of criteria such as consistency, relevance and minimality only partially addresses this issue. Many solutions within these criteria are difficult to obtain, particularly by means of the techniques described here.

The following section explains why the solutions that our algorithm generates are not semantically minimal.

5.4 Semantic minimality

Our algorithm only enforces the syntatic minimality of solutions (by means of the minimal hitting set algorithm), but not semantic minimality. This problem is best explained by means of an example.

Say we add the axiom $\exists$bloodTestIndicates.Plasmodium $\sqsubseteq$ $\exists$infectedWith.Malaria to the knowledge base of Example 1. Using the observation of Example 2, the algorithm now generates the solutions $\exists$infectedWith.Influenza(JOHN) and $\exists$bloodTestIndicates.Plasmodium(JOHN). One of the solutions we got previously, namely $\exists$infectedWith.Malaria(JOHN) has gone! In fact, a solution that is closer to semantic minimality has been lost.

Many such solutions that are closer to semantic minimality can be obtained by allowing expanded concept assertions as part of solutions (including the above example). However, this will not solve all problems: Consider the knowledge base consisting of TBox = $\{A_1 \sqcup A_2 \sqsubseteq A_3, \exists R.A_3 \sqsubseteq A_4\}$ and ABox = $\{R(I, J)\}$, and say we want abductive solutions for the observation $\{A_4(I)\}$. If we apply the algorithm to this problem, three solutions are generated: $\{A_1(J)\}$, $\{A_2(J)\}$ and $\{A_3(J)\}$. If we allow expandable assertions, we get $\{A_1 \sqcup A_2(J)\}$ and $\{\exists R.A_3(I)\}$ as solutions, but not $\{\exists R.A_1(I)\}$ or $\{\exists R.A_2(I)\}$. These are closer to semantic minimality than $\{\exists R.A_3(I)\}$.

Whether we manage to find a way of generating all semantically minimal solutions, or just those attainable by allowing expandable assertions, we imagine that the user of a system implementing an abduction algorithm would want to be able to explore a range of such solutions.

The notion of semantic minimality is related to the notion of weakest sufficient conditions [10], although this work is restricted to propositional logic. It is also reminiscent of work on least common subsumers [3], and we plan to investigate the possibility of applying those ideas to this situation.

6. CONCLUSION AND FUTURE WORK

In this paper, we have described how to implement ABox abduction in $\mathcal{ALC}$ directly in a DL tableau. Apart from the fact that it represents a simpler and more direct approach, Algorithm 2 does not implement many of the optimisations (e.g. back-jumping and caching) commonly used in DL tableau algorithms. Incorporating these into our algorithm promises to give a real efficiency advantage over the FOL connection tableau used in Klarman's algorithm.

This work also promises to be transferable to other more expressive DLs. The complications of dealing with role assertions (both in the observation and in the solutions) will disappear in languages that allow nominals, since negated role assertions can be expressed and reasoned about with nominals [8].

Languages that do not have the finite model property will need some means of dealing with infinite models. (We imagine that the current assertion set at the point of blocking could simply be added to the set of models collected by extendedST so that it will be closed by all solutions.)

As stated in Section 5.4, we also intend to investigate the work on weakest sufficient conditions and least common subsumers for their applicability to ranking solutions.

7. ACKNOWLEDGEMENTS

This work was partially funded by a European Union international research staff exchange scheme – Project number 247601, Net2: Network for Enabling Networked Knowledge, from the FP7-PEOPLE-2009-IRSES call. Thanks to Tommie Meyer of the CSIR Meraka Institute (in South Africa) and Enrico Franconi of the Free University of Bozen/Bolzano (in Italy) for infrastructure and support.

8. REFERENCES

[1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: *The Description Logic Handbook*,

Cambridge University Press, 2003.

[2] Baader, F., Horrocks, I., Sattler, U.: Chapter 3: Description Logics. In: van Harmelen, F., Lifschitz, V., Porter, B., editors: *Handbook of Knowledge Representation*, Elsevier, 2007.

[3] Baader, F., Sertkaya, B., Turhan, A.: Computing the least common subsumer w.r.t. a background terminology, *Journal of Applied Logic*, Springer, 2004.

[4] Di Noia, T., Di Sciascio, E., Donini, F.M.: Computing information minimal match explanations for logic-based matchmaking, in *Proc. of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, IEEE Computer Society, 2009.

[5] Du, J. Qi, G., Shen, Y-D., Pan, J.Z.: Towards practical ABox abduction in large OWL DL ontologies, in *Proc. of the 25th AAAI Conference*, 2011.

[6] Elsenbroich, C., Kutz, O., Sattler, U.: A case for abductive reasoning over ontologies, in *Proc. of the OWLED'06 Workshop*, vol 216, 2006.

[7] Halland, K., Britz, K.: Naïve Abox abduction in $\mathcal{ALC}$ using a DL tableau, in *Proc. of DL2012*, pp 443-453, 2012.

[8] Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$, in *Proc. of KR2006*, pp 57-67, 2006.

[9] Klarman, S., Endriss, U., Schlobach, S.: ABox abduction in the description logic $\mathcal{ALC}$, *Journal of Automated Reasoning*, vol 46:1, 2011.

[10] Lin, F.: On strongest necessary and weakest sufficient conditions, in *Proc. of KR2000*, pp 167-175, 2000.

[11] Reiter, R.: A theory of diagnosis from first principles, *Artificial Intelligence*, vol 32, 1987.

[12] Schild, K.: A correspondence theory for terminological logics: preliminary report. In *Proc. of IJCAI'91*, pp 446-471, 1991.

[13] Wotawa, F.: A variant of Reiter's hitting-set algorithm, *Information Processing Letters*, vol 79, 2001.