FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

OF THE COMENIUS UNIVERSITY IN BRATISLAVA

OPTIMIZATION OF AN ABDUCTIVE REASONER FOR DESCRIPTION LOGICS

Master thesis

2019 **Katarína Fabianová**

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
OF THE COMENIUS UNIVERSITY IN BRATISLAVA

OPTIMIZATION OF AN ABDUCTIVE REASONER FOR DESCRIPTION LOGICS

Master thesis

Study program:	Applied informatics
Field of study:	2511 Applied informatics
School department:	Department of Applied Informatics
Adviser:	Mgr. Júlia Pukancová, PhD.
Consultant:	RNDr. Martin Homola, PhD.

Bratislava 2019 **Katarína Fabianová**

Acknowledgements

Abstract

Key words:

Abstrakt

Kľúčové slová:

Contents

List of Figures

Introduction

Description logic

Description logics (DLs) are a family of knowledge representation. Each description logic has different expression. Every expression is expressed with a unique set of constructors. We are going to work with DL $\mathcal{EL}$ and DL $\mathcal{EL}$++. Later in this chapter we will introduce DL vocabulary and Ontology. DL vocabulary describes syntax and ontology describes semantics. Following definitions are from (Baader et al., 2008).

DL vocabulary

DL is dealing with individuals, concepts and roles. Individual is a concrete instance of a concept. It is used in *ABox* which will be defined later. Individuals are written with lower letters. Concept is written with first letter capital and the rest are lower letters. Concept can be atomic or complex. Atomic concept is not constructed with any constructor. On the contrary complex concept is constructed of other concepts. Complex concept is recursively defined as follows:

$$C, D ::= A|\neg C|C \sqcap D|C \sqcup D|\exists R.C|\forall R.C$$

In description logic we have also two concepts which are always in ontology. $\top$ (top) stays for everything. Each concept belongs under $\top$ which means that each concept is on left side of subsumption if on right side is only $\top$. Second concept is $\bot$ (bottom) and it stays for nothing which means that each concept is on the right side of subsumption if on left side is only $\bot$. Formally these two concepts can be written as:

$$\top \equiv A \sqcup \neg A$$

$$\bot \equiv A \sqcap \neg A$$

Role is a binary predicate in a form of restriction following some property and concept. Under restriction is meant: value restriction, existential restriction or number restriction. Property is written with first lower letter. Role can be written as follows:

$$\forall hasParent.Person$$

$$\exists owns.House$$

Description logic consists of three mutually disjoint sets. These sets represent whole domain that is used by DL.

$$\textbf{Set of individuals:} N_I = \{a, b, c...\}$$

$$\textbf{Set of concepts:} N_C = \{A, B, C...\}$$

$$\textbf{Set of roles:} N_R = \{R_1, R_2, R_3, ...\}$$

Description logic uses constructors such as $\neg$, $\sqcup$, $\sqcap$, $\exists$ and $\forall$. Constructor $\neg$ means negation, for instance $\neg Fruit$. Constructor $\sqcup$ means or, for instance $Mother \sqcup Father$. Constructor $\sqcap$ means and, for instance *Apple* $\sqcap$ *Pear*. Constructors $\exists$ is existential restriction and $\forall$ is called value restrictions. Among them exists also number restrictions but $\mathcal{EL}$ and $\mathcal{EL}$++ do not implement them.

Description logic uses this two important symbols: $\sqsubseteq$, $\equiv$. First symbol is subsumption $\sqsubseteq$. As an example we can consider expression $Mother \sqsubseteq Parent$. $Mother$ is always a *Parent* but *Parent* does not always have to be $Mother$. Second symbol is equivalence $\equiv$ which defines that both sides are equals. For example we can have expression *Parent* $\equiv Mother \sqcup Father$. *Parent* is always $Mother$ or $Father$. $Mother$ or $Father$ is also always *Parent*.

We will use following description logics: DL $\mathcal{EL}$ and DL $\mathcal{EL}$++. DL $\mathcal{EL}$ uses *TBox*, but no *ABox*. It does not use value restriction, only existential restriction. It also does not use disjunction but just conjunction. It uses also only atomic concepts or concepts of form $\top$. Roles are being used in form of existential restriction followed by property and concept. DL $\mathcal{EL}$++ is very similar to DL $\mathcal{EL}$ which means that uses everything what uses DL $\mathcal{EL}$ but includes also individuals so we can work with *ABox*.

To have a better understanding of how description logics works we will introduce a few examples. In all examples we will be rewriting following sentences into description logic.

Everybody who is sick, is not happy.

$$Sick \sqsubseteq \neg Happy$$

Cat and dogs are animals.

$$Cat \sqcup Dog \sqsubseteq Animal$$

Every person owns a house.

$$Person \sqsubseteq \exists owns.House$$

Ontology

Ontology describes relationships between entities in a specific area. Difference between area and ontology is that ontology describes relationships between entities in a formal language. In this part we are citing definitions from (Staab and Studer, 2010).

Every ontology has its own knowledge base. Knowledge base ($\mathcal{KB}$) is an ordered pair of *TBox* $\mathcal{T}$ and *ABox* $\mathcal{A}$, so we can write $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$. TBox represents all axioms that model ontology. Axiom has a form of predicate followed by operator and another predicate, for example $Apple \sqsubseteq Fruit$, $Person \sqsubseteq \exists own.House$. ABox on the other side does not model ontology but creates a database. It contains set of individuals. It has following form: individual : predicate. For example: $john : Person$, $greenApple : Apple$.

Definition 1.2.1 (Interpretation) *Interpretation of description logic is a pair* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *which contains a domain* $\Delta^{\mathcal{I}}$ *and an interpretation function* $\cdot^{\mathcal{I}}$.

Definition 1.2.2 (Domain) *Domain can not be empty. So it must be true that* $\Delta^{\mathcal{I}} \neq \emptyset$.

Definition 1.2.3 (Interpretation function) *Interpretation function is following:*

$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \; \forall a \in N_I$$

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \; \forall A \in N_C$$

$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \; \forall a \in N_R$$

Interpretation of complex concepts is recursively defined:

$$\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$$

$$C \sqcap D^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$

$$C \sqcup D^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$

$$\exists R.C^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \exists y \in \Delta^{\mathcal{I}} : \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$

$$\forall R.C^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \forall y \in \Delta^{\mathcal{I}} : \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$$

Interpretation is a model of knowledge base if $\mathcal{I}$ *satisfies every axiom in TBox and in ABox of given knowledge base.*

Ontology is for our work necessary because our input is always an ontology. We can create ontologies using an ontology modeling editor Protége or if we want we can also create an xml file. We have decided to use the editor where we created our ontologies. In the editor ontology can be saved in more types of xml files. We have used rdf/xml syntax.

DL Tableau algorithm

This algorithm is very important in our work because we are using it in ABox Abduction algorithm. DL Tableau algorithm proves satisfiability for an ABox axiom assertion. It means that we want to know if we are able to retrieve a model from ontology if we add some assertion axioms to ontology. We need to retrieve model from ontology in order to successfully continue in ABox abduction algorithm. Following definitions from (Baader et al., 2008) are describing rules and terms that are used by this algorithm.

Definition 1.3.1 (NNF) *A concept C is in NNF (negation normal form) only then if each negation constructor ($\neg$) is always before atomic concept symbols inside concept C.*

Example 1.3.1 (NNF) *Let's say that we have following axiom:*

$$\neg\forall owns.House \sqsubseteq Owner$$

We have to modify this axiom to be in NNF.

$$nnf(\neg\forall owns.House \sqsubseteq Owner) = \exists\neg owns.House \sqsubseteq Owner$$

Definition 1.3.2 (Completion tree) *A completion tree (CTree) is a triple T = (V, E, $\mathcal{L}$) where (V, E) is a tree and $\mathcal{L}$ is a labeling function which means that: $\mathcal{L}(x)$ is a set of concepts $\forall x \in V$ and $\mathcal{L}(\langle x, y\rangle)$ is a set of roles $\forall\langle x, y\rangle \in E$.*

Definition 1.3.3 (Successor, R-successor) *Given a CTree T = (V,E, $\mathcal{L}$) and $x, y \in V$ we say that: y is a successor of x if and only if $\langle x, y\rangle \in E$ and y is a R-successor of x if and only if $\langle x, y\rangle \in E$ and $R \in \mathcal{L}(\langle x, y\rangle)$.*

Definition 1.3.4 (Clash) *Clash is found in a CTree $T = (V, E, \mathcal{L})$ if and only if for some $x \in V$ and for some concept C both $C \in \mathcal{L}(x)$ and $\neg C \in \mathcal{L}(x)$.*

Definition 1.3.5 (CTree) *CTree is clash-free if and only if it is such a tree that in each branch has no clash.*

Definition 1.3.6 (Rules)

- $\sqcap - rule$: *if* $C_1 \sqcap C_2 \in \mathcal{L}(x)$ *and* $x \in V$ *and* $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$ *and* x *is not blocked then* $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

- $\sqcup - rule$: *if* $C_1 \sqcup C_2 \in \mathcal{L}(x)$ *and* $x \in V$ *and* $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ *and* x *is not blocked then either* $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1\}$ *or* $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$

- $\forall - rule$: *if* $\forall R.C \in \mathcal{L}(x)$ *and* $x, y \in V$ *and* y *is R-successor of* x *and* $C \notin \mathcal{L}(y)$ *and* x *is not blocked then* $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

- $\exists - rule$: *if* $\exists R.C \in \mathcal{L}(x)$ *and* $x \in V$ *with no R-successor* y *and* $C \in \mathcal{L}(y)$ *and* x *is not blocked then* $\mathcal{V} = \mathcal{V} \cup \{z\}, \mathcal{L}(z) = \{C\}$ *and* $\mathcal{L}(\langle x, z \rangle) = \{R\}$

- $\mathcal{T} - rule$: *if* $C_1 \sqsubseteq C_2 \in \mathcal{T}$ *and* $x \in V$ *and* $nnf(\neg C_1 \sqcup C_2) \notin \mathcal{L}(x)$ *and* x *is not blocked then* $\mathcal{L}(x) = \mathcal{L}(x) \cup \{nnf(\neg C_1 \sqcup C_2)\}$

Definition 1.3.7 (Blocking) *Given a CTree* $T = (V, E, \mathcal{L})$ *a node* $x \in V$ *is blocked if it has an ancestor* y *such that: either* $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ *or* y *is blocked.*

Example 1.3.2 ***(DL Tableau algorithm)*** *TODO add example na algorithm*

Abduction

Generally in logic we are familiar with three ways of thinking. Deduction, induction and abduction. The most known is probably deduction and for humans is most natural. All three ways are dealing with following parts: theory, data and effect. In description logic we can translate theory as knowledge base, data as explanations and effect as observation.

Deduction knows knowledge base and explanations, observation is missing and the goal is to deduce the missing observation. Induction knows observation and explanation but does not know knowledge base. Abduction knows knowledge base and observation but explanation is a subject of searching. All definitions are from article by (Pukancová and Homola).

ABox abduction

In description logic abduction is used when we are not familiar with explanation $\mathcal{E}$ but we know knowledge base $\mathcal{KB}$ and observation $\mathcal{O}$. It is important to know that we are looking for minimal explanations. Minimal explanation is such an explanation that does not exist any other explanation that would be a subset of this minimal explanation.

Definition 2.1.1 ***(Abduction)*** *Given knowledge base $\mathcal{KB}$ and observation $\mathcal{O}$, an abductive explanation is such explanation $\mathcal{E}$ that satisfies $\mathcal{KB} \cup \mathcal{O} \models \mathcal{E}$.*

Definition 2.1.2 ***(Correct explanation)***

$$\mathcal{E} \textit{ is consistent if } \mathcal{E} \cup \mathcal{KB} \not\models \bot;$$

$$\mathcal{E} \textit{ is relevant if } \mathcal{E} \not\models \mathcal{O};$$

$$\mathcal{E} \textit{ is explanatory if } \mathcal{KB} \not\models \mathcal{O}$$

Definition 2.1.3 ***(Minimal explanation)*** *Minimal explanation is such an explanation that does not exist other explanation that would be a subset of the first explanation.*

For better understanding of what ABox abduction is let's introduce a few examples. First example is easy, searched explanation is obvious but it will demonstrate the problem. Second example is not so obvious that's why we have to use an algorithm to compute the solution. For computing the solution we use **Minimal Hitting Set algorithm**.

Example 2.1.1 ***(ABox Abduction - Feeling)***

$$\mathcal{KB} = \left\{ \; Sick \sqsubseteq \neg Happy \; \right\}$$

$$\mathcal{O} = \left\{ \; mary : \neg Happy \; \right\}$$

In this example we are searching for explanations. In this easy assignment is obvious that what we are looking for is that Mary must be sick. If Mary is not happy she must be sick. Formally written solution to this abduction problem is following:

$$\mathcal{E} = \left\{ \; mary : Sick \; \right\}$$

Example 2.1.2 ***(ABox Abduction - Academy)***

$$\mathcal{KB} = \left\{ \begin{array}{l} Professor \sqcup Scientist \sqsubseteq Academician \\ AssocProfessor \sqsubseteq Professor \end{array} \right\}$$

$$\mathcal{O} = \left\{ \; jack : Academician \; \right\}$$

If *jack* is an Academician he must be Professor or Scientist. If he is an Professor he must be also an AssocProfessor. This is an oversimplified explanation how we can retrieve correct explanations. For better introduction into this algorithm we will explain it step by step. We know the observation so we know that jack is an Academician, that is a fact. In our first axiom in TBox is written that if somebody is an Academician he is also a Professor or Scientist. He can be both but at least one of them but that we are not able to determine. That is why are both correct explanations. The second axiom claims that if somebody is a Professor he must be also an AssocProfessor. In this part we are not explaining the whole algorithm yet because it will be explained in our next chapter Minimal Hitting Set algorithm but we need to have at least an idea of how does it work. So as we already know the correct explanations are following:

$$\mathcal{E} = \left\{ \begin{array}{l} jack : Professor \\ jack : Scientist \\ jack : AssocProfessor \end{array} \right\}$$

Minimal Hitting Set algorithm

In this part we will find out about algorithm: Minimal hitting set. This algorithm is invented by (Reiter, 1987). At first we will declare terms that we will be using.

Definition 2.2.1 (Hitting set) *Hitting set for a collection of sets C is a set $H \subseteq U_{S \in C}$ such that $H \cap S$ is not empty set for each $S \in C$.*

Definition 2.2.2 (HS-tree) *Let C be a collection of sets. An HS-tree T for C is a smallest edge-labeled and node-labeled tree with following properties:*

- *The root is labeled by ✓ if C is empty. Otherwise the root is labeled by an arbitrary set of C.*

- *For each node n of T, let $H(n)$ be the set of edge labels on the path in T from the root to node n. The label for n is any set $\sigma \in C$ such that $sigma \cap H(n) = \emptyset$, if such a set σ exists. Otherwise, the label for n is ✓. If n is labeled by the set σ, then for each $o \in \sigma$, n has a successor n_0 joined to n by an edge of labeled by o.*

Definition 2.2.3 (Generate pruned HS-tree)

- *Generate the pruned HS-tree breadth-first, generating all nodes at any fixed level in the tree before descending to generate the nodes at the next level.*

- *Reusing node labels: If node n has already been labeled by a set $S \in C$ and if n' is a new node such that $H(n') \cap S = \emptyset$, then label n' by S.*
- *Tree prunning:*
  - *If node n is labeled by ✓ and node n' is such that $H(n) \subseteq H(n')$, then close the node n'. A label is not computed for n' nor are any successor nodes generated.*
  - *If node n has been generated and node n' is such that $H(n') = H(n)$, then close node n'.*
  - *If nodes n and n' have been labeled by sets S and S' of C, respectively and if S' is a proper subset of S, then for each $\alpha \in S - S'$ mark as redundant the edge from node n labeled by α. A redundant edge, together with the subtree beneath it, may be removed from the HS-tree while preserving the property that the resulting pruned HS-tree will yield all minimal hitting sets for C.*

Reiter's algorithm is used to compute a minimal hitting set. Input is a set of sets. The goal of Reiter's algorithm is to compute minimal hitting set from this input. It means that the result hitting set will have intersection with each set from input.

Let's show an example with F as input set and HS as result.

Example 2.2.1 ***(Minimal Hitting Set)***

$$F = \{\{a,b\},\{b,c\},\{a,c\},\{b,d\},\{b\}\}$$

$$HS = \{\{a,b\},\{b,c\}\}$$

As we can observe in this example the first result set has intersection with each set of F and the second result set also has intersection with each set of F. So we can determine that these result sets are definitely minimal hitting sets. They are both minimal because there is no other set that is smaller and at the same time has intersection with each set of F.

For better and easier understanding of this algorithm we can visualize it and explain.

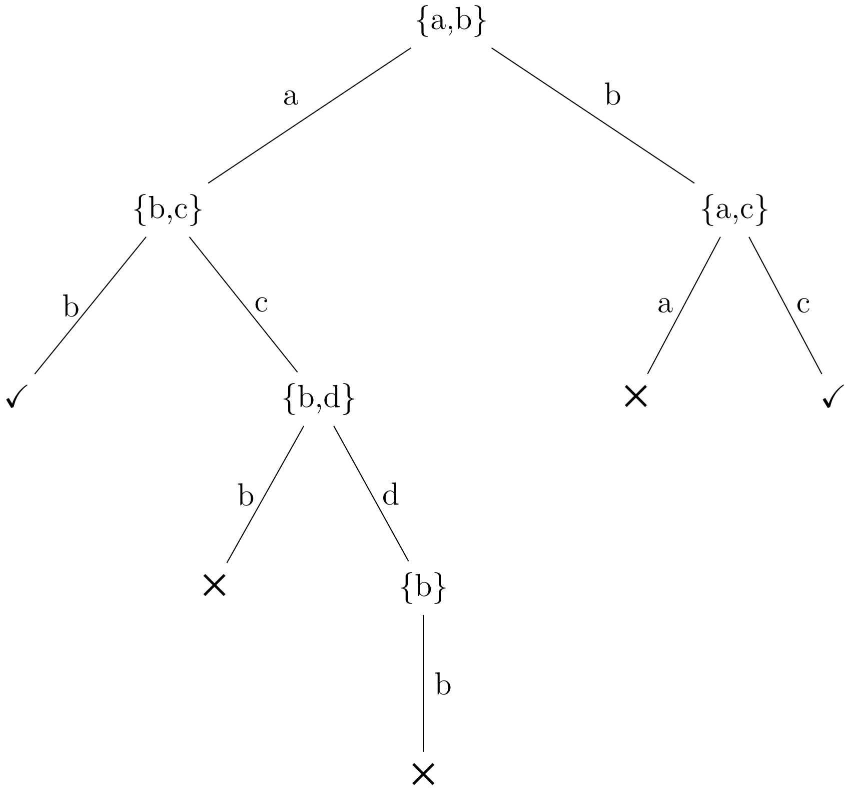

Figure 1: Minimal Hitting Set

Aim is to create a tree where nodes can have three types of value. Node can be set from F, check or cross. Each edge is labeled by one element of some set from F. At first we have to create root. Root node is first set $\{a, b\}$ from F. It will have exactly two children because the size of node is exactly two.

Now we have to decide what will be child's node. Let's check left child. We have to create a set from edges from root to our node. In this case this set contains only one element $\{a\}$. If $\{a\}$ has intersection with each sets from F we can add check as node. But that is not our case so we add the second set as node. Let's continue with right child of root. Labeled path from this node to root is $\{b\}$. That has also not intersection with each set of F so we add third set as node. We continue breadth-first with $\{b, c\}$ node. Left child has labeled path $\{a, b\}$ which has intersection with each set of F and it fulfills condition to be a possible minimal hitting set. Now we have to check if this possible minimal hitting set is really minimal. It is definitely a hitting set but we are looking for only minimal hitting sets. If already exists checked node that is a proper subset of this possible hitting set we have to add cross as node because it is not a minimal hitting set, otherwise we add check as node. Similarly we continue in this algorithm and we get two minimal hitting sets: $\{a, b\}$ and $\{b, c\}$.

Optimizations

To Reiter's original algorithm were added optimizations by (Greiner et al., 1989) and (Wotawa, 2001) a few years later.

Greiner claims that Reiter's way of creating a tree handles some situations incorrect. The base algorithm is correct but prunning may lead to loose of minimal hitting sets. Proposed way of preventing this loose is use directed acyclic graph instead of a tree. We did not use this optimization because in our case wrong handling of situation can not happened.

The next optimization by Wotawa disclaim the proposed way by Greiner. He has returned to using a tree. He tries to save time and make the algorithm quicker. His idea is to create deterministic tree where nodes are sorted from left to right where left means the smallest size of edge label and right means the biggest size of edge label in one breadth level. In our case this optimization is also useless because in our tree in one breadth level the size of edge label is always equal.

Reasoning

Elk reasoner

Other reasoners

Implementation

ABox Abduction with Minimal Hitting Set algorithm

For our algorithm to work we have to combine these two algorithms as designed by (Pukancová and Homola).

TODO add pseudocode

MergeXPlain algorithm

TODO add pseudocode

ABox Abduction algorithm

TODO add pseudocode

Conclusion

References

Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. *Foundations of Artificial Intelligence*, 3:135–179, 2008.

Steffen Staab and Rudi Studer. *Handbook on ontologies*. Springer Science & Business Media, 2010.

Júlia Pukancová and Martin Homola. Tableau-Based ABox Abduction for the $\mathcal{ALCHO}$ Description Logic.

Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32 (1):57–95, 1987.

Russell Greiner, Barbara A Smith, and Ralph W Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

Franz Wotawa. A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001.

Appendices