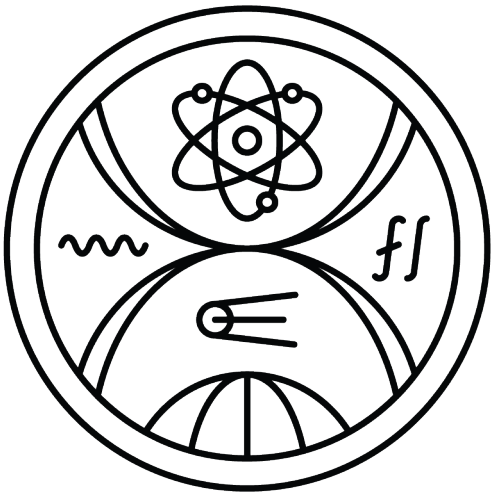
COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

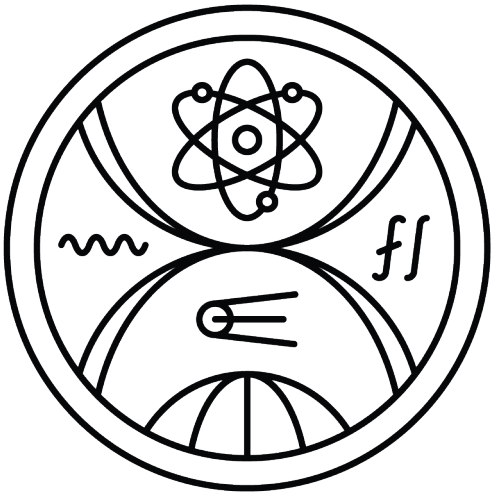


OPTIMIZATION OF THE MHS-MXP ALGORITHM

Master thesis

2024 Jakub Kloc

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS



OPTIMIZATION OF THE MHS-MXP ALGORITHM

Master thesis

Study Program:	Applied Computer Science
Field of Study:	Computer Science
School Department:	Department of Applied Informatics
Supervisor:	doc. RNDr. Martin Homola, PhD.
Consultant:	mgr. Janka Boborová

Bratislava, 2024

Jakub Kloc

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Jakub Kloc
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Optimization of the MHS-MXP algorithm

Annotation: The MHS-MXP abduction algorithm improves MHS by applying a divide-and-conquer strategy (MXP) to search through a space of possible explanations (HS-tree). The MXP runs are iterated, therefore suitable search heuristics, tree-pruning, and caching may possibly be used to improve the execution time in consecutive iterations.

Aim: Evaluate the existing implementation of the MHS-MXP algorithm. Propose and implement improvements in search strategy, memory use, caching, etc. Evaluate the efficiency on a suitable test case.

Literature:

1. Elsenbroich, C., Kutz, O., Sattler, U., 2006. A case for abductive reasoning over ontologies. In: OWLED
2. Reiter, R., 1987. A theory of diagnosis from first principles. Artificial intelligence, 32(1):57-95
2. Shchekotykhin, K., Jannach, D., Schmitz, T., 2015. MergeXplain: Fast computation of multiple conflicts for diagnosis. In IJCAI
3. Homola, M., Pukancová, J., Boborová, J. and Balintová, I., 2023. Merge, explain, iterate: A combination of MHS and MXP in an ABox abduction solver. In: JELIA

Supervisor:	doc. RNDr. Martin Homola, PhD.
Consultant:	Mgr. Janka Boborová
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	11.12.2023
Approved:	12.12.2023

prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Kloc
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Optimization of the MHS-MXP algorithm
Optimalizácia algoritmu MHS-MXP

Anotácia: Abduktívny algoritmus MHS-MXP využíva metódu rozdeľuj a panuj (MXP) pri prehľadávaní priestoru možných vysvetlení (HS-strom). Behy MXP sú iterované, preto môže vhodná heuristika, orezávanie stromu a kešovanie informácií z predchádzajúcich behov potenciálne zlepšiť čas behu v nasledujúcich iteráciách.

Cieľ: Kriticky vyhodnotiť súčasnú implementáciu algoritmu MHS-MXP. Navrhnúť možné zlepšenia na úrovni prehľadávacej stratégie, práce s pamäťou, kešovania, a pod. Evalvovať efektívnosť zlepšení na vhodne zvolených testovacích dátach.

Literatúra:
1. Elsenbroich, C., Kutz, O., Sattler, U., 2006. A case for abductive reasoning over ontologies. In: OWLED
2. Reiter, R., 1987. A theory of diagnosis from first principles. Artificial intelligence, 32(1):57-95
2. Shchekotykhin, K., Jannach, D., Schmitz, T., 2015. MergeXplain: Fast computation of multiple conflicts for diagnosis. In IJCAI
3. Homola, M., Pukancová, J., Boborová, J. and Balintová, I., 2023. Merge, explain, iterate: A combination of MHS and MXP in an ABox abduction solver. In: JELIA

Vedúci: doc. RNDr. Martin Homola, PhD.
Konzultant: Mgr. Janka Boborová
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 11.12.2023

Dátum schválenia: 12.12.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstract

Abstract

Keywords:

Abstrakt

Abstrakt

Kľúčové slová:

Contents

Introduction

Part I

State of the art

Chapter 1

Description Logics

Description logics [3] (DLs) are a family of logic languages used in knowledge representation. Their purpose is to describe some domain of interest by reducing it into an abstraction that consists of standalone elements, often called individuals. An individual (which can be any distinguishable entity, not just an individual person or object) may be place into categories or form relationships with other individuals. In this chapter, we will describe the exact syntax and semantics of DLs.

1.1 Syntax

There are many different DLs, usually defined by adding additional expressivity onto some existing DL [3]. By convention, these languages are then named by letters representing which features they contain. One of the smallest DLs that still provide a meaningful range of features is called $\mathcal{ALC}$ [2]. We will describe $\mathcal{ALC}$ to demonstrate the basic syntax and semantics that are included in most DLs.

As already mentioned, DLs describe *individuals* [3] that belong to some domain. The individuals can be put into sets called concepts. A *concept* [3], identified by a concept name, may represent some category of elements or a trait that is possessed by each individual in the set. This includes two special concepts - $\top$ (all elements of the domain, pronounced top) and $\bot$ (an empty

set, pronounced bottom).

Similar to concepts, *roles* [3] are sets of couples and represent a relationship between two elements. They are identified by role names.

Definition 1.1.1 (Vocabulary) *A **vocabulary** of a DL domain, also called its signature, consists of three countable, mutually disjoint sets: **individual names** N_I, **concept names** N_C and **role names** N_R.*

Apart from these names, a DL language also contains operators, often called *constructors* [3], that allow us to combine names and represent more complex ideas. In table 1.1, we provide an overview of constructors in $\mathcal{ALC}$. We can then distinguish between *atomic* (those that are represented by a single concept name) and *complex* (those that are build using constructors) *concept expressions* [3, 19].

Symbol	Name	Example	Intuitive meaning
$\neg$	negation	$\neg$Human	those who are not humans
$\sqcap$	conjunction	Animal $\sqcap$ Pet	animals that are also pets
$\sqcup$	disjunction	Human $\sqcup$ Animal	those who are either people or animals
$\exists$	existential restriction	$\exists$hasPet.Dog	those who have a pet dog
$\forall$	value restriction	$\forall$hasPet.Dog	those who only have dogs as pets

Table 1.1: $\mathcal{ALC}$ constructors

Definition 1.1.2 (Atomic concept expression) *An **atomic concept expression** is an expression formed by the following grammar:*

$A ::= C \mid \top \mid \bot$,

where $C \in N_C$.

Definition 1.1.3 (Complex concept expression) *A **complex concept expression** is an expression formed by the following grammar:*

$D, E ::= A \mid \neg D \mid D \sqcap E \mid D \sqcup E \mid \forall r.D \mid \exists r.D,$
where A *is an atomic concept expression and* $r \in N_R$.

Definition 1.1.4 (Concept expression) *A* ***concept expression*** *is any atomic or complex concept expression.*

Now that we defined the basic syntactic elements, let us look at how they are used to represent knowledge. A single statement is called an axiom. Axioms are grouped into a collection called a knowledge base (KB) [3, 2], consisting of two parts: a TBox and an ABox.

A TBox contains information about the knowledge base's terminology, i.e. general rules about concepts and their relationship. An equality axiom states that two concepts are equivalent. Its main usage is to define a new concept by equaling it with a complex concept expression. Another type of a TBox axiom is so called general concept inclusion (GCI) axiom. It states that each element from a concept C must also be included in another concept D. This allows us to define hierarchies of concepts. An equality axiom $C \equiv D$ is effectively a shortcut for two CGI axioms, $C \sqsubseteq D$ and $D \sqsubseteq C$.

Definition 1.1.5 (TBox axioms) *A* ***TBox axiom*** *is an expression of the form:*
$\alpha ::= C \equiv D$ *(****equality*** *axiom) or*
$\beta ::= C \sqsubseteq D$ *(****GCI*** *axiom),*
where C, D *are concept expressions.*

Definition 1.1.6 (TBox) *A* ***TBox*** T *is a set of TBox axioms.*

A knowledge base's Abox contains assertional information about specific elements from the domain. Its axioms either declare that an individual belongs to a concept, or that a couple of individuals belongs to a role.

One may notice that $\mathcal{ALC}$ does not provide the same level of expressivity for roles as it does for concepts. For our practical purposes, it will be beneficial to be able to negate a role assertion, similarly to a concept assertion with negated concept name. Despite it not being an explicit part of $\mathcal{ALC}$, we will include it in our definition as well.

Definition 1.1.7 (ABox axioms) *An **ABox axiom** is an expression of the form:*

$\alpha ::= C(i)$ *(**concept assertion**),*

$\beta ::= r(i,j)$ *(**role assertion**) or*

$\gamma ::= \neg r(i,j)$ *(**negated role assertion**),*

where C is a concept expression, $r \in N_R$ and $i, j \in N_i$.

Definition 1.1.8 (ABox) *An **ABox** A is a set of ABox axioms.*

Definition 1.1.9 (Knowledge base) *A knowledge base K is a set of axioms A T.*

1.2 Semantics

To be able to understand a KB's meaning, we need to interpret it w.r.t. a concrete domain, using a structure called an *interpretation* [3, 19]. In an interpretation, the domain of interest is represented as a set of elements. It is not necessarily required to be finite. An *interpretation function* maps each individual name to an element it corresponds to. For each concept expression, it returns all the elements that belong to the concept (this is called the concept's *extension*), and for each role name it returns every couple of elements that are in that binary relationship (the role's extension).

Definition 1.2.1 (Interpretation) *An **interpretation** $\mathcal{I}$ of a knowledge base is a pair $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, where **domain** $\Delta^{\mathcal{I}}$ is a non-empty set of elements and **interpretation function** $.^{\mathcal{I}}$ is a function that provides the following mapping:*

$$
\begin{gathered}
i^{\mathcal{I}} \in \Delta^{\mathcal{I}}, \\
C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \\
r^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}), \\
\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} = \emptyset, \\
\neg D^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}},
\end{gathered}
$$

$$D \sqcap E^{\mathcal{I}} = D^{\mathcal{I}} \cap E^{\mathcal{I}},$$
$$D \sqcup E^{\mathcal{I}} = D^{\mathcal{I}} \cup E^{\mathcal{I}},$$
$$\exists r.D^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}} \wedge y^{\mathcal{I}} \in D^{\mathcal{I}}\},$$
$$\forall r.D^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}} \implies y^{\mathcal{I}} \in D^{\mathcal{I}}\},$$

where $i \in N_I$, $C \in N_C$, $r \in N_R$ and D, E are concept expressions.

Using an interpretation, it is possible to evaluate whether a KB axiom is true in the given domain, i.e. if it is *satisfied* [3].

Definition 1.2.2 (Axiom satisfaction) *An axiom α is satisfied in an interpretation $\mathcal{I}$ ($\mathcal{I} \models \alpha$), if one of the following holds true:*

$$\alpha \text{ is } D \equiv E \text{ and } D^{\mathcal{I}} = E^{\mathcal{I}},$$
$$\alpha \text{ is } D \sqsubseteq E \text{ and } D^{\mathcal{I}} \subseteq E^{\mathcal{I}},$$
$$\alpha \text{ is } D(i) \text{ and } i^{\mathcal{I}} \in D^{\mathcal{I}},$$
$$\alpha \text{ is } r(i,j) \text{ and } (i^{\mathcal{I}}, j^{\mathcal{I}}) \in r^{\mathcal{I}},$$
$$\alpha \text{ is } \neg r(i,j) \text{ and } (i^{\mathcal{I}}, j^{\mathcal{I}}) \notin r^{\mathcal{I}},$$

where $i, j \in N_I$, $r \in N_R$ and D, E are concept expressions.

When we understand how to evaluate the truth value of an axiom, it is possible to take an axiom and construct a new one that has an opposite truth value in every interpretation. We will call axioms like these *negated axioms* and limit them to an ABox setting, as that will be the main focus in our work.

Definition 1.2.3 (Negated ABox axiom) *For an Abox axiom α, we can construct a **negated axiom** $\neg\alpha$ according to the following table. For any interpretation $\mathcal{I}$, $\mathcal{I} \models \alpha$ if and only if $\mathcal{I} \not\models \neg\alpha$.*

α	$\neg\alpha$
$D(i)$	$\neg D(i)$
$\neg D(i)$	$D(i)$
$r(i,j)$	$\neg r(i,j)$
$\neg r(i,j)$	$r(i,j)$

where $i, j \in N_I$, $r \in N_R$ and D is a concept expressions.

An interpretation which satisfies each axiom from a KB is called the KB's *model*. It represents a state of the domain which exactly corresponds with the knowledge in the KB. However, it is important to realize that an interpretation can contain any number of elements and mappings that are not affected by the KB at all. A single model of a KB can be modified into an infinite number of new interpretations by simply adding any arbitrary mappings that do not clash with the KB [19].

Definition 1.2.4 (Model) *An interpretation $\mathcal{I}$ is a **model** of a knowledge base K, noted $\mathcal{I} \models K$, if for each axiom $\alpha \in K$ it holds that $\mathcal{I} \models \alpha$.*

When practically working with models in a programming setting, it is useful to not have to store the whole interpretation. For the purposes of our work, we will define a reduced representation of a model. It contains Abox axioms that are satisfied in the model and use only the vocabulary of the KB's domain, ignoring irrelevant elements and mappings. Moreover, the assertion axioms will be limited to only contain atomic concept expressions, concept expressions with negated concept names and their role counterparts. These are sufficient to describe the mappings in the interpretation. This reduction will yield the same result for each model that interprets the KB equally.

1.3 Reasoning

A major purpose of representing knowledge formally is to be able to reason over it. A most typical reasoning task is to infer data that is implicitly included as a logical consequence of the explicitly stated data. In a DL KB, if an axiom is true in each of its models, it is the KB's logical consequence - it is *entailed* by it [19, 2].

Definition 1.3.1 (Axiom entailment) *A knowledge base K **entails** an axiom α, noted $K \models \alpha$, if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models K$, it holds that $\mathcal{I} \models \alpha$.*

However, to infer data from a KB, we need to assure that the knowledge it represents is internally consistent. A consistent KB is such that it has at least one model. If no model exists for a KB, it means that its axioms can never be all true at the same time - the knowledge they represent is contradictory with itself [19].

Definition 1.3.2 (Consistent knowledge base) *A knowledge base K is **consistent** if there is at least interpretation $\mathcal{I}$ such that $\mathcal{I} \models K$. Otherwise, it is **inconsistent**.*

For an axiom to be entailed by a KB, it must be satisfied in each of its models. If the numbers of models is zero, no axiom breaks the condition, as there is never a model that would not satisfy it. As a consequence, an inconsistent KB entails every axiom and thus does not provide any meaningful information.

Lemma 1.3.1 (Reasoning over inconsistent knowledge base) *Let K be an inconsistent knowledge base. For any axiom α, $K \models \alpha$.*

There is a situation where it is desirable to encounter an inconsistent KB. To check if an ABox axiom α is entailed by some KB K, going by definition, we would need to check all of K's models for axiom satisfaction. There is a more efficient way, however: If we negate α and add it to K, if the resulting knowledge base $K' = K \cup \neg\alpha$ has any model $\mathcal{I}$, that model breaks the condition of entailment: $\mathcal{I} \models \neg\alpha$, thus $\mathcal{I} \not\models \alpha$, and α cannot be entailed. On the other hand, if K' has no model, then α's negation can never be true in the same interpretation as K, and as a result, α must be true in every model of K, thus K entails it [19].

Lemma 1.3.2 (Reducing ABox entailment to consistency) *A knowledge base K entails an axiom α ($K \models \alpha$) if and only if $K \cup \{\neg\alpha\}$ is inconsistent.*

Chapter 2

Abduction

Abduction is a method of inference that tries to explain how some observation could be entailed by existing knowledge. In this chapter, we will first briefly introduce its philosophical background, then formalize it in DL an explain how to filter and limit desirable explanations.

2.1 Abductive reasoning

Abduction has been declared as a standalone type of inference along the more well-known deduction and induction by scientist and philosopher Charles Peirce [4, 18] . While deduction directly infers logical consequences of facts and induction builds general rules from existing examples, abduction aims to learn new facts that could explain the reason behind some observed phenomenon. In other words, it reasons backwards from a consequence to find its cause. However, it is hypothetical - it can arrive to a lot of different conclusions and does not assure which one corresponds to reality [18].

In computer science, applications of abduction have been studied in many domains, such as natural language processing, planning problems, deductive databases, knowledge repairing, multimedia interpretation, engineering diagnosis or medical diagnosis [9, 10, 1, 16].

Formally, the input of an abduction problem consists of two components: *background knowledge* (what we already know) and an *observation* (what

we are trying to explain). The background knowledge does not entail the observation, but we are trying to change that by adding new facts into it - *explanations* [4, 10].

2.2 DL ABox abduction

In the context of DLs, there are multiple different problems that are based on abductive reasoning [4, 10]. We will focus on ABox abduction, trying to extend the ABox to explain an ABox axiom.

Definition 2.2.1 (ABox abduction problem) *An **ABox abduction problem** is a couple $(\mathcal{K}, \mathcal{O})$, where the **background knowledge** $\mathcal{K}$ is a DL knowledge base and the **observation** $\mathcal{O}$ is an Abox axiom such that $\mathcal{K} \cup \{\mathcal{O}\}$ is consistent. The solution for this problem is a set of Abox axioms $\mathcal{E}$, an **explanation**, such that $\mathcal{K} \cup \mathcal{E} \models \mathcal{O}$.*

Not every set of axioms that matches the previous definition of an explanation has to be valid as a solution to a real-life problem. To obtain only meaningful explanations, we can limit them by applying multiple criteria, described in 2.2 [4, 10].

Definition 2.2.2 (Explanation types) *An explanation of an abduction problem $(\mathcal{K}, \mathcal{O})$ is:*

- ***consistent** if $\mathcal{K} \cup \mathcal{E}$ is consistent*
- ***relevant** if $\mathcal{E} \not\models \mathcal{O}$*
- ***explanatory** if $\mathcal{K} \not\models \mathcal{O}$*
- ***minimal** if there is no other explanation $\mathcal{X}$ of $(\mathcal{K}, \mathcal{O})$ such that $\mathcal{X} \subseteq \mathcal{E}$*

Consistency is the most important requirement, as according to 1.3, an inconsistent KB entails every axiom. That means that any axiom breaking $\mathcal{K}$'s consistency is automatically an explanation for any observation. Just

like any reasoning made on the resulting $\mathcal{K}$', an explanation like this would not make any practical sense.

Relevancy assures that the observation is not a trivial consequence of an explanation by itself.

Explanatoriness describes the abduction problem as a whole. If the background knowledge already entails the observation, there is no need to solve the problem at all. Any axiom added into $\mathcal{K}$ without breaking its consistency would be considered an "explanation", even though it does not explain anything.

As for *minimality*, it is often a desired condition, but it may be defined in multiple different ways [4, 10]. The general idea is that if we find multiple explanations that are fully interchangeable, we want to choose only the one that is somehow the "smallest" of them. In our case, we will define an explanation's size by the number of axioms in it.

Definition 2.2.3 (Explanation size) *The size of an explanation (a set of axioms) $\mathcal{E}$ is the number of axioms in it.*

Just like with models, by taking an explanation and adding any number of arbitrary axioms to it that would not break any of our requirements, we could get to an infinite number of explanations that are practically the same from the problem's perspective. We prevent this by eliminating explanations that are supersets of other explanations.

Another method of limiting the explanations is choosing only specific vocabulary that can be included in them, e.g. only specific concepts or specific individuals. The allowed vocabulary is called the *abducibles* [8]. In the end, any restriction like this can be formulated as a set of axioms that are allowed to be included in the explanations.

Taking abducibles into consideration, we can redefine the abduction problem, as even if we do not desire any limitation, the abducibles can simply be a set of all possible axioms that can exist in the vocabulary.

Definition 2.2.4 (ABox abduction problem with abducibles) *An ABox abduction problem is a couple $(\mathcal{K}, \mathcal{O}, Abd)$, where the **background knowledge** $\mathcal{K}$ is a DL knowledge base, the **observation** $\mathcal{O}$ is an Abox axiom such*

that $\mathcal{K} \cup \{\mathcal{O}\}$ *is consistent, and* ***abducibles*** *Abd is a set of Abox axioms. The solution for this problem is a set of Abox axioms* $\mathcal{E}$*, an* ***explanation****, such that* $\mathcal{K} \cup \mathcal{E} \models \mathcal{O}$ *and* $\mathcal{E} \subseteq Abd$*.*

Chapter 3

Abduction algorithms

In this chapter, we will introduce multiple algorithms capable of solving abduction problems and finding all possible explanations. We will start with Reiter's algorithm for finding diagnoses of a faulty system [17]. While it was not originally designed with DL abduction in mind, it has been successfully adapted to solve this problem [14]. The following algorithms, HST [20] and Rc-Tree [12], are heavily based on Reiter's algorithm to improve its functionality. Even though they have not yet been used in DL abduction tools, it is logical to assume they can be adapted the same way.

3.1 Reiter's MHS algorithm

Reiter's work [17] concerned diagnostic reasoning systems. In this context, a system may be any real-life domain consisting of individual components interacting with each other, described by an abstraction in some logical language. When we observe that such a system doesn't work correctly, a diagnosis identifies which components caused it by behaving abnormally. Reiter proved that finding all diagnoses for a faulty system can be achieved by an algorithm looking for *minimal hitting sets*, and designed such an algorithm.

For some tuple of sets F, a hitting set is one that contains at least one element from each of the sets included in F. For it to be minimal, it can't be a subset of any other hitting set for F.

Despite its original purpose, the algorithm is abstract enough to be used with various logic formalisms and solve problems other than system diagnosis [17, 6, 14]. As we further explain the algorithm, we will substitute all terms and actions in the algorithm that relate to diagnoses with those that are relevant for DL abduction (later, we will also apply the same principles to all the algorithms derived from this one). As the algorithm lacks a proper official name, from now on, we will call it by by a name used in abduction-related works [6, 8], the *Minimal Hitting Set algorithm* (MHS).

MHS works by constructing a tree structure called a hitting set tree (HS-tree)[17, 14]. Both the vertices and edges in the tree can be labeled - the edges with axioms, the vertices either with a model, a ✕ mark or a ✓ mark (we will explain these marks later).

For each vertex v, the algorithm also defines a function $h(v)$, that looks at the sequence of edges leading from the tree's root to v (we will sometimes refer to it as v's *path*), and returns a set of axioms that label these edges.

Algorithm 3.1.1 (MHS) *The MHS algorithm has the following steps:*

1. *Initialize the HS-tree by creating a root vertex.*

2. *Process every unprocessed vertex in the tree in BFS order. A vertex v is processed as follows:*

   (a) *A consistency check is performed on $\mathcal{K} \cup \neg O \cup h(v)$. If the union is inconsistent, $h(v)$ is an explanation. In that case, mark v with ✓ and skip the remaining steps (the node is closed).*

   (b) *If the union was consistent, then it must have some model $\mathcal{I}$. Acquire $\mathcal{I}$'s ABox representation M and create its negated version N. Label v with N.*

   (c) *For each axiom α in N, create an edge leading from v to a newly created vertex v'. The edge is labeled with the axiom α.*

3. *As soon as there are no vertices left to process, the HS-tree is finished. Each $h(v)$ of a vertex v marked with ✓ is an explanation.*

One way how to look at the algorithm, independently from the concept of hitting sets, is that it attempts to "break" the models by negating their axioms and finding the one negation that finally makes the knowledge base inconsistent: If $\mathcal{K} \cup \neg O \cup h(v)$ is still consistent, taking some axiom from the resulting model and negating it will have a higher chance of being a conflict than adding any random axiom into the knowledge base.

Note that for the tree's root r the function $h(r)$ returns an empty set , as there are no edges from the root to itself. If $\mathcal{K} \cup \neg O \cup$ is inconsistent, the observation already entails from $\mathcal{K}$ and there is no need to explain anything. This breaks the requirement of explanatoriness and makes the abduction problem itself unnecessary.

Closing of vertices that found an explanation helps to assure minimality, along with the BFS strategy. In the first level of the tree, all paths have size one, so if any size one explanation exists, it is found in this level. Adding any more axioms to that path would produce only non-minimal explanations by definition, so there is no meaning in continuing the branch and the vertex is closed.

However, this doesn't prevent a non-minimal explanation from being found in another branch of the tree where we generated some superset of $h(v)$ by adding axioms in a different order.

To help with this and other possible redundant branches, the algorithm also includes three pruning conditions [17]:

1. If there is an unprocessed vertex v and another vertex v' that is labeled with ✓ so that $h(v') \subseteq h(v)$, close v. It will not be processed nor will $h(v)$ be considered an explanation, as it would be non-minimal. Mark it with ✗ to indicate it's closed, but not an explanation.

2. If there is an unprocessed vertex v and another vertex v' such that $h(v') = h(v)$, close v with ✗.

3. If there are vertices v and v' that have been respectively labeled by negated models N and N' such that $N \subseteq N'$, then for each axiom α

from $N' \setminus N$, prune the edge coming from v labeled by α and the whole subtree below it, removing it from the tree.

The third condition is based on Reiter's observation that if two models are found and one of them is a subset of the other, the subset is all we need to find a minimal hitting set [17]. The remaining axioms from the larger model are thus redundant and we can prune their subtrees without losing any explanations.

The ability to decrease the HS-tree's size is important, as it can grow immensely when processing large inputs. Another way how to prevent it is to limit the tree's depth. Thanks to the BFS strategy, if we only desire explanations up to size of some x, we can stop the algorithm after xth level of the tree is finished.

3.2 HS-Dag and RC-Tree

After Reiter's work was published, an error has been discovered in it [5]: by themselves, the pruning conditions are correct. However, if all of them are used at once, the completeness of the algorithm is not assured. The first two conditions rely on the assumption that a vertex can be closed, as a branch that would lead to the same explanation(s) already exists. However, let us consider the following situation: If some vertex v is closed because it is redundant w.r.t. some other vertex v', and v' gets later pruned by the third condition, the possible explanations that would be found under either of these vertices are entirely lost from the final tree.

A revision of the algorithm, called HS-DAG, was proposed [5]. Instead of a HS-tree, it constructs a directional acyclic graph (DAG), where if two equivalent paths would be created, instead of the second one being closed, it points to the same vertex v as the first one. If one of v's parents is pruned, v still remains connected to the graph by the other path and explanations under it are not lost.

HS-DAG itself was later improved upon by the *RC-tree* algorithm [12], standing for "HS-tree with reduced conflicts". This algorithm builds on HS-

DAG and extends it with other features that prevent redundant branches from ever being created, while still assuring that no explanations are lost. This allows the data structure to remain as a tree instead of a DAG.

Apart from $h(v)$, Rc-Tree defines a property of vertices $\Theta(v)$, that stores a set of axioms. These axioms cannot used as labels for the edges leading from v to its children.

Algorithm 3.2.1 (Rc-Tree) *The Rc-Tree algorithm has the following steps:*

1. *Initialize the tree by creating a root vertex r with $\Theta(r) = \varnothing$.*
2. *Process every unprocessed vertex in the tree in BFS order. A vertex v is processed as follows:*
   (a) *A consistency check is performed on $\mathcal{K} \cup \neg O \cup h(v)$. If the union is inconsistent, $h(v)$ is an explanation. In that case, mark v with ✓ and skip the remaining steps (the node is closed).*
   (b) *If the union was consistent, then it must have some model $\mathcal{I}$. Acquire $\mathcal{I}$'s ABox representation M and create its negated version N. Label v with N.*
   (c) *For each axiom α in $N \setminus \Theta(v)$, create an edge leading from v to a newly created vertex v'. The edge is labeled with the axiom α. $\Theta(v')$ will inherit all axioms from $\Theta(v)$. It will also include all axioms that have at this point been used to label an edge from v, including α.*
3. *As soon as there are no vertices left to process, the tree is finished. Each $h(v)$ of a vertex v marked with ✓ is an explanation.*

We can see that as the algorithm progresses, Θs of the vertices are being filled with axioms that have already been used as labels in other branches, not only in the vertex's current level, but in previous levels as well. This assures that every branch generated by the tree will have a unique combination of axioms on its edges.

Even more important than the Θs, however, are changes made to the pruning condition. The second of the original three is no longer needed, as the tree cannot generate the same path twice. The third one has been hugely modified to not only prune redundant branches, but to create new ones when necessary to prevent loss of explanations.

The pruning conditions are as follows:

1. If there is an unprocessed vertex v and another vertex v' that is labeled with ✓ so that $h(v') \subseteq h(v)$, close v with ✗.

2. After a vertex v was labeled by a negated model N (in step 2a of the algorithm), for every vertex v' in the tree, if it is labeled by some N' such that $N \subset N'$, attempt to prune the tree as follows:

   (a) Relabel v' with the smaller model N.

   (b) For each axiom α from $N' \setminus N$ (that is, for each axiom that is no longer in the label of v'), prune the edge coming from v' labeled by α and remove its subtree. (This may remove the original v itself from the tree. In that case, after the pruning is finished, step 2b of the algorithm is skipped for v).

   (c) For each child v'' of v', set $\Theta(v'')$ to $\Theta(v'') \setminus \{\alpha\}$ and propagate the change to update to all descendants of v''.

   (d) For each vertex v''' which had α removed from its Θ, create a new edge labeled by α leading from v''' to a newly created vertex.

3.3 HST

Another algorithm that with the purpose to fix MHS's shortcomings is the Hitting Set Tree algorithm [20] (HST; note that the structure built by these algorithms is also called a hitting set tree, but these are two separate terms). More specifically, it was designed to reduce the number of subset checks performed.

Just like RC-Tree, it aims to prevent redundant branches from ever being created, but it uses a different strategy: Where RC-Tree explicitly remembers which edges have already been created, HST systematically generates every single possible combination of axioms that have so far appeared in found models. The method is inspired by an algorithm that finds all subsets of a given set.

In HST, each vertex v has two integer values associated with it: an index $i(v)$ and a minimum $min(v)$ (we will explain their meaning later). Also, v can store an information about which vertex is its parent using a function $parent(v)$ and which vertices are its children using a function $child(v, n)$, where n is an integer. It returns the child that is connected to v by an edge by an axiom with index n - axioms are indexed as well.

More precisely, each axiom α that has already appeared in some model is given an integer index $ai(\alpha)$. The indices start at the maximal value (total number of abducibles) and decrease from there. The next index to be assigned to an axiom is stored in a global variable Min.

This means that edges in a HST tree can be labeled by the indices instead of axioms themselves. However, the labeling has been moved to the vertices, with the vertex index $i(v)$ serving that purpose. As a consequence, the function $h(v)$ has been redefined to return axioms that share indices with the vertices on the path from the root to v.

Algorithm 3.3.1 (HST) *The HST algorithm has the following steps:*

1. *Set Min to be the number of abducibles.*
2. *Initialize the tree by creating a root vertex r with $i(r) = Min + 1$ and $parent(r) = none$.*
3. *Process every unprocessed vertex in the tree in BFS order. A vertex v is processed as follows:*
   (a) *A consistency check is performed on $\mathcal{K} \cup \neg O \cup h(v)$. If the union is inconsistent, $h(v)$ is an explanation. In that case, mark v with ✓ and stop processing the vertex.*

(b) *If the union was consistent, then it must have some model* $\mathcal{I}$*. Acquire* $\mathcal{I}$*'s ABox representation* M *and create its negated version* N*. Label* v *with* N*.*

(c) *For every axiom* α *in* N *such that* $ai(\alpha)$ *is not defined, set* $ai(\alpha)$ *to* Min *and decrease* Min *by one.*

(d) *Set* $min(v)$ *to* $Min+1$*. If* $i(v) \leq min(v)$*, close* v *with* ✗ *and stop processing the vertex.*

(e) *For each integer* n *in the range* $\langle\ min(v)\ ;\ i(v)-1\ \rangle$*, create a new vertex* v' *with* $i(v') = n$*. Set* $child(v, n) = v'$ *and* $parent(v') = v$*.*

4. *As soon as there are no vertices left to process, the tree is finished. Each* $h(v)$ *of a vertex* v *marked with* ✓ *is an explanation.*

We can see that when children for a vertex v should be created, instead of choosing the axioms in the model labeling v, HST always generates axioms represented by a strictly given integer range: bounded by the values of $min(v)$ and $i(v)$.

$min(v)$ is derived from the global value of the smallest index that has yet appeared. If an axiom from abducibles has never appeared in any negated model, there is no need to try it, as by definition, it cannot belong to a minimal hitting set of negated models.

$i(v)$ is assigned similarly to Θ in Rc-Tree, as it is always smaller than the index of v's parent, but larger than the indices of its other siblings on the same level.

Together, these bounds ensure that an edge for every relevant axiom is created, except for those that would be redundant w.r.t to the current subtree.

HST has only one pruning condition to be checked during the algorithm's run (note that with the original third condition removed, HST succesfully removes the need to perform subset checks on the found models):

1. If there is an unprocessed vertex v and another vertex v' that is labeled with ✓ so that $h(v') \subset h(v)$, close v with ✗.

However, additional pruning is proposed after the tree has been finished: To prune each node closed by ✗ and remove it from the list of its parent's children (redefine the result of $child(v, n)$). Afterwards, if some non-✓ vertex is left without any children, it is removed as well. This recursively repeats until we are left with a tree whose each branch represents some explanation.

Chapter 4

MHS-MXP Solver

4.1 MHS-MXP algorithm

4.2 Solver usage

4.3 Optimisations

Part II

Our contribution

Chapter 5

Implementation of new algorithms

5.1 Refactoring

5.2 HST

5.3 RC-Tree

5.4 QXP and MXP

Chapter 6

Optimisations

6.1 Model extraction changes

Chapter 7

Evaluation

In this chapter, we will describe how we evaluated the performance of the algorithms in the CATS solver. We will go over the main goals we focused on during the evaluation, the metrics we measured, the input ontologies and observations we chose, and finally over the results themselves and our interpretation of them.

7.1 Goals

Our main motivation was to find strengths and weaknesses of the different approaches and compare them, as there were very few comparisons of that many abduction algorithms at once [13, 11], and, to our knowledge, no mass comparison of abduction algorithms working with DL ontologies.

We were also especially interested in the overall traits of the algorithms we introduced for the first time, HST-MXP and RCT-MXP. If some of them could reliably out-perform the MHS-MXP algorithm, it would be a likely candidate to become the new "default" algorithm of CATS.

7.2 Choosing the metrics

Previously, the MHS-MXP solver's performance has been evaluated based on the average time when a specific level of the HS-tree was finished [7], thus

focusing mainly on the speed of the algorithms. There are, however, different approaches, e.g. to measure how many vertices were created and pruned [15] or how much of the system memory was used by the running process [13]. For this evaluation, we implemented counters that keep track of most of the basic actions that happen during each algorithm's run. For each level of the HS-tree, we store the following data:

- number of vertices that were processed (that were already created in the previous level)
- number of edges that were created from the processed vertices, how many of them were closed with ✓ and how many were closed with ✗
- number of nodes that were created
- how many from the created vertices re-used a model as their label
- how many model extractions and consistency checks were performed
- how many vertices were deleted by Rc-Tree, differentiating those that were deleted before being processed
- the times when the level started (the first vertex with that depth was polled from a queue) and when it ended (the first vertex with the next depth was polled)
- the times when the first and the last explanations were found in that level

7.3 Choosing the inputs

Conclusion

Conclusion

Bibliography

[1] C. Alrabbaa, S. Borgwardt, T. Friese, P. Koopmann, and M. Kotlov. Why not? explaining missing entailments with evee. In O. Kutz, C. Lutz, and A. Ozaki, editors, *Proceedings of the 36th International Workshop on Description Logics (DL 2023) co-located with the 20th International Conference on Principles of Knowledge Representation and Reasoning and the 21st International Workshop on Non-Monotonic Reasoning (KR 2023 and NMR 2023)., Rhodes, Greece, September 2-4, 2023*, volume 3515 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[4] C. Elsenbroich, O. Kutz, and U. Sattler. A case for abductive reasoning over ontologies. CEUR-WS.org, 2006.

[5] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artif. Intell.*, 41(1):79–88, 1989.

[6] K. Halland and K. Britz. Abox abduction in *ALC* using a DL tableau. In J. H. Kroeze and R. de Villiers, editors, *2012 South African Institute of Computer Scientists and Information Technologists Conference,*

SAICSIT '12, Pretoria, South Africa, October 1-3, 2012, pages 51–58. ACM, 2012.

[7] M. Homola, J. Pukancová, I. Balintová, and J. Boborová. Hybrid MHS-MXP abox abduction solver: First empirical results. CEUR-WS.org, 2022.

[8] M. Homola, J. Pukancová, J. Gablíková, and K. Fabianová. Merge, explain, iterate. In S. Borgwardt and T. Meyer, editors, *Proceedings of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020*, volume 2663 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[9] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6), 12 1992.

[10] S. Klarman, U. Endriss, and S. Schlobach. Abox abduction in the description logic *ALC*. *J. Autom. Reason.*, 46(1):43–80, 2011.

[11] R. Koitz-Hristov and F. Wotawa. Faster horn diagnosis - a performance comparison of abductive reasoning algorithms. *Appl. Intell.*, 50(5):1558–1572, 2020.

[12] I. Pill and T. Quaritsch. Rc-tree: A variant avoiding all the redundancy in reiter's minimal hitting set algorithm. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Gaithersburg, MD, USA, November 2-5, 2015*, pages 78–84. IEEE Computer Society, 2015.

[13] I. Pill, T. Quaritsch, and F. Wotawa. On the practical performance of minimal hitting set algorithms from a diagnostic perspective. 7(2), 2016.

[14] J. Pukancová and M. Homola. Tableau-based abox abduction for the ALCHO description logic. In A. Artale, B. Glimm, and R. Kontchakov,

editors, *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017*, volume 1879 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[15] J. Pukancová and M. Homola. Abox abduction for description logics: The case of multiple observations. In M. Ortiz and T. Schneider, editors, *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018*, volume 2211 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[16] S. M. Rashid, J. McCusker, D. Gruen, O. Seneviratne, and D. L. McGuinness. A concise ontology to support research on complex, multimodal clinical reasoning. Berlin, Heidelberg, 2023. Springer-Verlag.

[17] R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[18] C. T. Rodrigues. The method of scientific discovery in peirce's philosophy: Deduction, induction, and abduction. *Logica Universalis*, 5(1):127–164, 2011.

[19] S. Rudolph. Foundations of description logics. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. F. Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 76–136. Springer, 2011.

[20] F. Wotawa. A variant of reiter's hitting-set algorithm. *Inf. Process. Lett.*, 79(1):45–51, 2001.

List of Figures