TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Theoretical Computer Science, Chair for Automata Theory

Efficient reasoning for lightweight temporal description logics

Timon Bertold Hoschke

Born on: 21.08.2002 in Erfurt
Matriculation number: 5037642

Bachelor's Thesis

to achieve the academic degree

Bachelor of Science (B.Sc.)

First referee
Dr.-Ing. Stefan Borgwardt

Second referee
Prof. Dr. Markus Krötzsch

Submitted on: 06.08.2024

Statement of authorship

I hereby certify that I have authored this document entitled *Efficient reasoning for lightweight temporal description logics* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 06.08.2024

Timon Bertold Hoschke

Abstract

Many real-world processes, like the weather, are time-dependent, a dimension to data that is not widely supported in ontology-based representations. Therefore, this thesis is dedicated to implementing the reasoning algorithm from Borgwardt, Forkel, and Kovtunova, which decides entailment for $\mathcal{TELH}_{\perp}^{\Diamond,\mathbf{lhs}}$, a lightweight and tractable temporal description logic, in the Datalog-based rule engine *Nemo*. The implementation works on normalised ontologies specified in a temporalised sublanguage of *OWL 2 EL*. Furthermore, the implementation is evaluated on an ontology using real-world weather data, which shows promising results but also suggests specific optimisations to make reasoning viable on large datasets.

Contents

1 Introduction

Basing query-answering technologies for ontology-based data on description logics (DLs) offers a crucial advantage over traditional databases: the ability to work with incomplete data. While databases are expected to contain all the information needed to answer queries, ontologies can use reasoning algorithms to infer missing information. Thus, it might only be necessary to provide some basic facts and the background knowledge to derive the needed new information.

In applications, data is often not only incomplete but also time-dependent: the weather changes, stock prices fluctuate, and a patient's health evolves. This promising prospect has led to high interest in temporalising existing DLs [3, 17] and obtaining tractable reasoning procedures [2]. Temporal DLs extend classical DLs with temporal operators, often from linear temporal logic (LTL) [2, 9]. Thus, they allow the modelling of temporal data by associating facts with time points at which they are valid. For example, $givesBirth(cee, dee, 2002)$ specifies that cee gave birth to dee in 2002 and the axiom $\Diamond\!\!-\!\exists givesBirth.\top \sqsubseteq BiologicalMother$ states that someone who gave birth sometime in the past is a biological mother. Although this is already a powerful formalism, it does not yet allow for the modelling of the distance of events. For example, $\Diamond\!\!-\!HasFeaver \sqsubseteq Sick$ is quite useless, as it states that someone who has had a fever in the past will be sick from then on till the end of time. A more reasonable axiom would be to specify that someone is sick between two measurements of a raised body temperature, which are at most three days apart. To be able to make such statements, Borgwardt, Forkel, and Kovtunova [5] proposed $\mathcal{TELH}_{\bot}^{\Diamond\!\!-,\mathsf{lhs}}$, a lightweight temporal DL with metric, convex temporal operators. In this DL, the example can be formulated as $\Diamond\!\!-_3 HasFeaver \sqsubseteq Sick$. It is, therefore, also possible to work with incomplete information in the temporal dimension.

Borgwardt, Forkel, and Kovtunova [5] propose a reasoning algorithm for $\mathcal{TELH}_{\bot}^{\Diamond\!\!-,\mathsf{lhs}}$ which decides entailment, but this procedure has yet to be implemented. Thus, as a step towards making queries on temporal ontologies possible, this thesis implements the reasoning algorithm for $\mathcal{TELH}_{\bot}^{\Diamond\!\!-,\mathsf{lhs}}$ in the modern Datalog-based rule engine *Nemo* and evaluates its performance. First, the necessary background knowledge is provided in Chapter 2. It includes explanations of $\mathcal{TELH}_{\bot}^{\Diamond\!\!-,\mathsf{lhs}}$ and the temporal operators it is based upon, as well as the completion algorithm itself. On the more technical side, Section 2.4 and Section 2.5 introduce the *Resource Description Framework* (*RDF*) and the *OWL 2 Web Ontology Language* (*OWL 2*), a standard for describing ontologies, and the rule engine *Nemo*, respectively. Second, Chapter 3 describes the implementation of the completion algorithm in *Nemo*, including preprocessing steps and the translation of the algorithm's rules. Following that, Chapter 4 discusses the implementation's performance. The algorithm is benchmarked using an ontology-based on real-world weather data. Finally, Chapter 5 concludes this thesis by summarising the results and discussing possible future work.

2 Preliminaries

This chapter first introduces the metric linear-time temporal logic operators mentioned in the introduction. Then, based on them, the temporal description logic $\mathcal{TELH}_{\bot}^{\Diamond_{\mathsf{lhs}}^{\diamond}}$ is defined. Third, the completion algorithm for $\mathcal{TELH}_{\bot}^{\Diamond_{\mathsf{lhs}}^{\diamond}}$ is presented, and a short overview of OWL is given. Next, Datalog and the rule engine *Nemo* are introduced. Finally, the chapter concludes with a discussion of the implementation of another reasoner in *Nemo*.

2.1 $\Diamond_n^{\diamond}$-Operators: Metric Linear-Time Temporal Logic Operators

This section introduces the metric linear-time temporal logic operators, which are later used to define the description logic $\mathcal{TELH}_{\bot}^{\Diamond_{\mathsf{lhs}}^{\diamond}}$. LTL formulae are constructed over a finite set P of *propositional variables*. As in Borgwardt, Forkel, and Kovtunova [5], this section only considers formulae generated from the grammar rule $\varphi ::= p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond_I \varphi$, where $p \in P$, and I is an interval in $\mathbb{Z}$. Negation and other logical connectives like implications do not need to be taken into account, because $\mathcal{TELH}_{\bot}^{\Diamond_{\mathsf{lhs}}^{\diamond}}$ does not permit them. Therefore, they are irrelevant to this thesis. Disjunction is the only exception, as it is necessary to define the metric convex diamond operator. An LTL formula can be satisfied by an infinite sequence $\mathfrak{W} = (w_i)_{i \in \mathbb{Z}}$, where $w_i \subseteq P$, called an *LTL-structure*. Informally, each w_i represents a time point, and p occurring in w_i denotes p as true at that moment. More formally, the semantics is defined as follows [5]:

$$\mathfrak{W}, i \models p \text{ iff } p \in w_i, \qquad \mathfrak{W}, i \models \varphi \wedge \psi \text{ iff } \mathfrak{W}, i \models \varphi \text{ and } \mathfrak{W}, i \models \psi,$$
$$\mathfrak{W}, i \models \Diamond_I \varphi \text{ iff } \exists k \in I : \mathfrak{W}, i + k \models \varphi, \qquad \mathfrak{W}, i \models \varphi \vee \psi \text{ iff } \mathfrak{W}, i \models \varphi \text{ or } \mathfrak{W}, i \models \psi$$

In order to clarify the following definitions, a few derived operators are introduced with $n \geq 1$ [5]:

$$\boxplus \varphi := \Diamond_{(-\infty,\infty)} \varphi \qquad \boxplus \varphi := \Diamond_{[0,\infty)} \varphi \qquad \boxminus \varphi := \Diamond_{(-\infty,0]} \varphi$$
$$\boxdot \varphi := \Diamond_{(-\infty,0]} \varphi \wedge \Diamond_{[0,\infty)} \varphi \qquad \boxdot_n \varphi := \bigvee_{\substack{k,m \geq 0 \\ k+m=n-1}} (\Diamond_{[-k,0]} \varphi \wedge \Diamond_{[0,m]} \varphi)$$

The $\boxplus$ operator conveys the notion that φ will be true at some point in the future. Thus, this operator is also known as the "eventually" operator in classical LTL. The variant $\boxminus$ requires φ to be true at some point in the past, while $\boxplus$ requires that φ is true at some point in the past *or* the future. The operator $\boxdot$ can be used to express a convex closure of time points, as it necessitates that φ must be true at some point in the past *and* the future. Finally, the operators $\boxdot_n$ denote a metric variant of $\boxdot$, stipulating that distinct occurrences of φ must be separated by no more than $n - 1$ time points, thereby enclosing an interval of length n.

For a more straightforward analysis of these operators, it is helpful to focus on the effect a diamond operator has when applied to a single propositional variable p. Thus, consider the set of time points at which $\Diamond^{\star} p$ holds when given the set of time points at which p is true. $\Diamond^{\star}$ acts as a placeholder for any one of the previously defined operators. In the following, $\Diamond^{\bullet}$, $\Diamond^{\dagger}$, and $\Diamond^{\ddagger}$ will be used similarly as placeholders for different diamond operators. The usefulness of the following properties will become apparent when discussing the efficiency of the reasoning algorithm.

Definition 1 ([5]). *Consider the sets* $\mathfrak{D}^{\mathsf{c}} := \{\Diamond^{c}\} \cup \{\Diamond^{c}_i \mid i \geq 1\}$, $\mathfrak{D}^{\pm} := \{\Diamond^{+}, \Diamond^{-}, \Diamond^{\pm}\}$, *and* $\mathfrak{D} := \mathfrak{D}^{\pm} \cup \mathfrak{D}^{\mathsf{c}}$ *of* diamond operators. *Each* $\Diamond^{\star} \in \mathfrak{D}$ *induces a function* $\Diamond^{\star} : 2^{\mathbb{Z}} \to 2^{\mathbb{Z}}$ *with* $\Diamond^{\star}(M) := \{i \mid \mathfrak{W}_M, i \models \Diamond^{\star} p\}$ *for all* $M \subseteq \mathbb{Z}$ *, with the LTL-structure* $\mathfrak{W}_M := (w_i)_{i \in \mathbb{Z}}$ *such that* $w_i := \{p\}$ *if* $i \in M$, *and* $w_i := \emptyset$ *otherwise.*

As indicated above, M represents a set of time points at which p is true and $\Diamond^{\star}(M)$ produces the set of time points at which $\Diamond^{\star} p$ holds. In the following, the parenthesis in $\Diamond^{\star}(M)$ will be omitted for a cleaner presentation. Notice, if M is empty, $\Diamond^{\star} M$ is empty as well, for any $\Diamond^{\star} \in \mathfrak{D}$. For any non-empty $M \subseteq \mathbb{Z}$, the following expressions are obtained, where max M is the greatest number in M and min M the smallest. Thus, max M represents the furthest point in the future at which p holds, and min M is the furthest point in the past. It is possible that max M is ∞ and min M is $-\infty$.

$$\Diamond^{\pm} M = \mathbb{Z} \qquad \Diamond^{+} M = (-\infty, \max M] \qquad \Diamond^{-} M = [\min M, \infty) \qquad \Diamond^{c} M = [\min M, \max M]$$
$$\Diamond^{c}_1 M = M \qquad \Diamond^{c}_{\mathsf{n}} M = \{i \in \mathbb{Z} \mid \exists j, k \in M \text{ with } j \leq i \leq k \text{ and } k - j < n\}$$

The effect of some diamond operators is shown in Figure 1. There, the timeline marked with $\mathbb{Z}$ represents the set of all time points. As an example, p is true at -3, -2, 0, and 4. Thus, $M = \{-3, -2, 0, 4\} = \Diamond^{c}_1 M$, which is illustrated with red dots on the $\mathbb{Z}$-time line. Each following row visualises a set $\Diamond^{\star} M$ for a different $\Diamond^{\star} \in \mathfrak{D}$ noted on the left, where each dot depicts an element in $\Diamond^{\star} M$. Intervals of length > 1, within which $\Diamond^{\star} p$ is inferred to hold, are marked with $\vdash\!\!\!-\!\!\!-\!\!\!\dashv$. Intervals of length 1 are not explicitly shown, as $M \subseteq \Diamond^{\star} M$ always holds. For example, $\Diamond^{c}_3 M$ is the set of time points at which p is true in intervals of length 3 or less. These intervals are $[-3, -2]$, $[-2, 0]$, and $[i, i]$ with $i \in M$. Thus, $\Diamond^{c}_3 M = \{-3, -2, -1, 0, 4\}$.

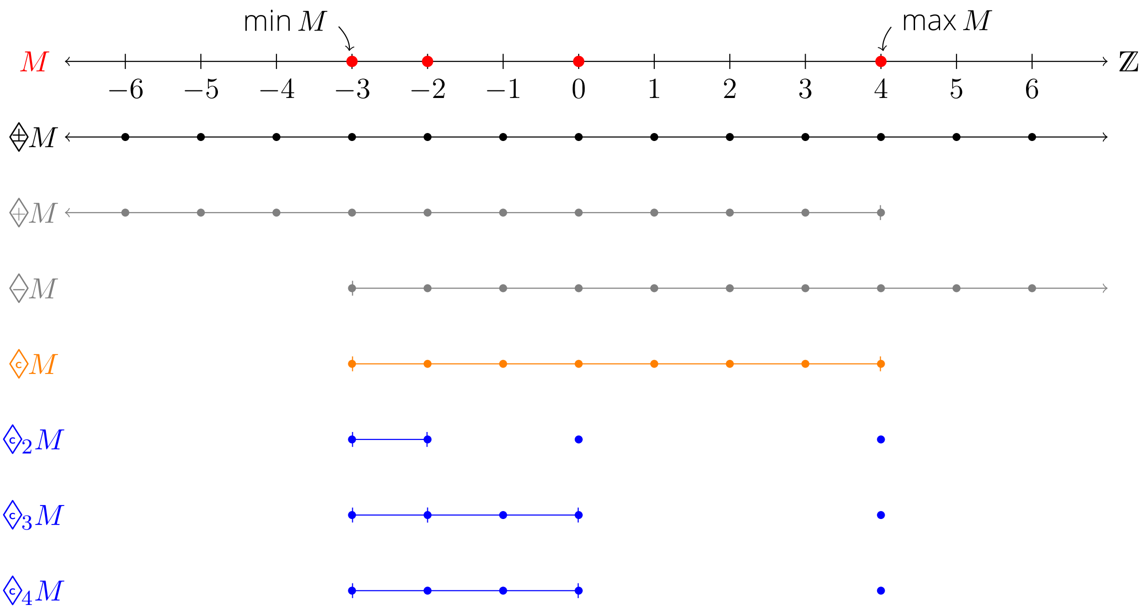
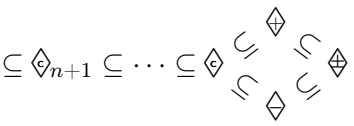


Figure 1: A graphic representation of $\Diamond^{\star} M$ for different $\Diamond^{\star} \in \mathfrak{D}$ with $M = \{-3, -2, 0, 4\}$

Lemma 2 ([5]). *The following ordered set $(\mathfrak{D}, \subseteq)$, where $id_{2^{\mathbb{Z}}}$ is the identity function on $2^{\mathbb{Z}}$, is obtained using the pointwise inclusion order $\subseteq$ on the induced:*

$$id_{2^{\mathbb{Z}}} = \Diamond_1 \subseteq \cdots \subseteq \Diamond_n \subseteq \Diamond_{n+1} \subseteq \cdots \subseteq \Diamond$$



Lemma 3 ([5]). *The set $\mathfrak{D}$ is closed under composition $\circ$, pointwise intersection $\cap$, and pointwise union $\cup$, and for any $\Diamond, \blacklozenge \in \mathfrak{D}$ these operators can be computed as:*

$$\Diamond \cap \blacklozenge = \inf_{(\mathfrak{D},\subseteq)}\{\Diamond, \blacklozenge\} \qquad \textit{and} \qquad \Diamond \circ \blacklozenge = \Diamond \cup \blacklozenge = \sup_{(\mathfrak{D},\subseteq)}\{\Diamond, \blacklozenge\}$$

where $\inf_{(\mathfrak{D},\subseteq)}$ denotes the infimum in $(\mathfrak{D}, \subseteq)$, and $\sup_{(\mathfrak{D},\subseteq)}$ the supremum.

Thus, notice that diamond operators can be combined by $\circ$, $\cap$, and $\cup$ without leaving the set $\mathfrak{D}$.

2.2 $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$: A Lightweight Temporal Extension of $\mathcal{ELH}_{\bot}$

This section defines the temporal description logic $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ using the diamond operators from above. The reader is assumed to be familiar with *description logics* (DLs). Otherwise, for a general introduction to DLs, one may refer to [16].

Syntax. Let N_C, N_R, and N_I be disjoint sets of *concept*, *role*, and *individual names*, respectively. $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ concepts are built upon atomic concepts and roles, which, in turn, are recursively combined to form complex concepts, as seen in Table 1, where C and D denote concepts, $A \in N_C$, $r, s \in N_R$, $a \in N_I$, and $\Diamond \in \mathfrak{D}$. If a concept C does not contain subexpressions of the form $\Diamond D$, it is syntactically an $\mathcal{ELH}_{\bot}$ concept and is therefore referred to as an *atemporal concept*.

A $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ ontology $\mathcal{O}$ is a finite set of axioms divided into a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A TBox is comprised of *concept inclusions* (CIs) $C \sqsubseteq D$ and *role inclusions* (RIs) $r \sqsubseteq s$, where C is a $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ concept, D is an atemporal concept, r is a temporal role, and $s \in N_R$. D must be an atemporal concept, as allowing the occurrence of diamond operators on the right-hand side of a CI would make the logic undecidable [2]. The concept equivalence $C \equiv D$ is an abbreviation for the CIs $C \sqsubseteq D$ and $D \sqsubseteq C$. Similarly, for RIs. An ABox includes *concept assertions* $A(a, i)$ and *role assertions* $r(a, b, i)$, where $A \in N_C$, $r \in N_R$, $a, b \in N_I$, and $i \in \mathbb{Z}$.

Semantics. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *domain* $\Delta^{\mathcal{I}} \supseteq N_I$ and an *interpretation function* $\cdot^{\mathcal{I}}$. This function assigns to each atomic concept $A \in N_C$ a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to each individual a an element $a^{\mathcal{I}}$ from $\Delta^{\mathcal{I}}$, and to each atomic role $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Such an interpretation, of course, does not include a temporal dimension and, therefore, needs to be modified as follows. A *temporal interpretation* $\mathcal{J} = (\Delta^{\mathcal{J}}, (\mathcal{I}_i)_{i\in\mathbb{Z}})$ consists of a domain $\Delta^{\mathcal{J}}$ and a series $(\mathcal{I}_i)_{i\in\mathbb{Z}}$ of interpretations $\mathcal{I}_i = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{I}_i})$ with $i \in \mathbb{Z}$. Intuitively, each of these interpretations represents the current "state" of the concepts/roles at each time point i. The formal definition of the semantics can be seen in Table 1.

$\mathcal{J}$ *satisfies* a CI $C \sqsubseteq D$ if for all $i \in \mathbb{Z}$, $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$ holds. Likewise, a RI $r \sqsubseteq s$ is satisfied by $\mathcal{J}$ if $r^{\mathcal{I}_i} \subseteq s^{\mathcal{I}_i}$ holds for all $i \in \mathbb{Z}$. $\mathcal{J}$ satisfies a concept assertion $A(a, i)$ if $a \in A^{\mathcal{I}_i}$ holds and similarly, a role assertion $r(a, b, i)$ if $(a, b) \in r^{\mathcal{I}_i}$ holds. In general, $\mathcal{J}$ satisfying an axiom α is written as $\mathcal{J} \models \alpha$. If $\mathcal{J}$ satisfies all axioms in an ontology $\mathcal{O}$, $\mathcal{J}$ is called a *model* of $\mathcal{O}$ (written $\mathcal{J} \models \mathcal{O}$). An ontology $\mathcal{O}$ is *consistent* if it has a model, and it *entails* α (written $\mathcal{O} \models \alpha$) if all models of $\mathcal{O}$ satisfy α. An *inconsistent* ontology $\mathcal{O}$ has no models, which is true iff $\mathcal{O} \models \top \sqsubseteq \bot$. Therefore, the completion algorithm from Borgwardt, Forkel, and Kovtunova [5] implemented in this paper focuses on deciding entailment analogous to reasoners for similar DLs [13]. In $\mathcal{ELH}_{\bot}$ [4] as well as $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ [5], this is a tractable decision problem.

The semantics for temporal roles $(\Diamond_\dagger r)^{\mathcal{I}_i}$ and concepts $(\Diamond_\dagger C)^{\mathcal{I}_i}$ might seem rather convoluted, but they are actually significantly more intuitive than they appear. First, note that the current time point is i. Second, observe that the set $\{j \mid d \in C^{\mathcal{I}_i}\}$, which will be denoted as N, is almost identical to M from Definition 1. M consists of time points at which a propositional variable is true, and N does essentially the same in the context of DLs, as it collects the time points j at which an element d is in $C^{\mathcal{I}_j}$. Thus, $\Diamond_\dagger N$ is comprised of all the time points at which d is in $\Diamond_\dagger C$. Now, the condition $i \in \Diamond_\dagger N$ checks whether d is currently in $\Diamond_\dagger C$ or not. Thus, one could imagine applying a diamond operator to a concept as a figurative "stretching effect" on occurrences of elements along the time axis. So, the occurrence of element d in $C^{\mathcal{I}_i}$ is inferred ("stretched out") to other time points through the application of $\Diamond_\dagger$ to C. For example, if d currently satisfies C, it will, from now on, always satisfy $\Diamond_\dagger C$ in the future. Its occurrence got "stretched out" to every future time point. Thus, the concept $(\Diamond_\dagger C)^{\mathcal{I}_i}$ contains all elements currently in $C^{\mathcal{I}_i}$ in addition to all the elements that were at some point in the past in C (see Figure 1). The interpretation function $\cdot^{\mathcal{I}_i}$ works analogously for temporal roles.

The algorithm requires an ontology's CIs/RIs to be in the normal form:

$$\Diamond_\dagger A \sqsubseteq B,\ A_1 \sqcap A_2 \sqsubseteq B,\ \Diamond_\dagger r \sqsubseteq s,\ \Diamond_\dagger A \sqsubseteq \exists r.B,\ \exists r.A \sqsubseteq B,$$

where $\Diamond_\dagger \in \mathfrak{D}$, $A, A_1, A_2, B \in N_C \cup \{\top, \bot\}$, and $r, s \in N_R$. However, this normalisation comes without loss of generality, as complex concepts can be simulated by introducing fresh concept and role names as abbreviations. For example, $(\Diamond_\dagger A) \sqcap B \sqcap C \sqsubseteq D$ can be simulated by $\Diamond_\dagger A \sqsubseteq A'$, $B \sqcap C \sqsubseteq E$ and $A' \sqcap E \sqsubseteq D$.

Axioms consisting only of atemporal concepts and roles such as $A \sqsubseteq B$ or $r \sqsubseteq s$ can be treated as their equivalent temporal counterparts $\Diamond_1 A \sqsubseteq B$ and $\Diamond_1 r \sqsubseteq s$, respectively.

Given Lemma 2, the notion of a *unique strongest* axiom entailed by an ontology $\mathcal{O}$ is easily obtained. It is the axiom $\Diamond_\dagger A \sqsubseteq B \in \mathcal{O}$ with $\Diamond_\dagger \circ \Diamond_\bullet = \Diamond_\dagger$, for any other $\Diamond_\bullet A \sqsubseteq B \in \mathcal{O}$. For example, $\mathcal{O} \models \Diamond_\forall A \sqsubseteq B$ implies $\mathcal{O} \models \Diamond_\dagger A \sqsubseteq B$ and $\mathcal{O} \models \Diamond_\exists A \sqsubseteq B$, as $\Diamond_\dagger A \sqsubseteq \Diamond_\forall A$ and $\Diamond_\exists A \sqsubseteq \Diamond_\forall A$. Thus, $\Diamond_\forall A \sqsubseteq B$ is the unique strongest axiom and $\Diamond_\exists A \sqsubseteq B$ and $\Diamond_\dagger A \sqsubseteq B$ are redundant in this case. This notion will become important when discussing redundant entailments made by the algorithm.

2.3 A Completion Algorithm for $\mathcal{TELH}_\bot^{\Diamond_\exists,\mathsf{lhs}}$

This section presents the reasoning algorithm from Borgwardt, Forkel, and Kovtunova [5], the implementation of which will be discussed in Chapter 3. The procedure works by repeatedly

Table 1: Syntax and semantics of $\mathcal{TELH}_\bot^{\Diamond_\exists,\mathsf{lhs}}$

	Syntax	Semantics
Roles:		
Atomic role	r	$r^{\mathcal{I}_i} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$
Temporal role	$\Diamond_\dagger r$	$\{(d,e) \in \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}} \mid i \in \Diamond_\dagger\{j \mid (d,e) \in r^{\mathcal{I}_j}\}\}$
Concepts:		
Atomic concept	A	$A^{\mathcal{I}_i} \subseteq \Delta^{\mathcal{J}}$
Temporal concept	$\Diamond_\dagger C$	$\{d \in \Delta^{\mathcal{J}} \mid i \in \Diamond_\dagger\{j \mid d \in C^{\mathcal{I}_j}\}\}$
Top	$\top$	$\Delta^{\mathcal{J}}$
Bottom	$\bot$	$\emptyset$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}_i} \cap D^{\mathcal{I}_i}$
Existential restriction	$\exists r.C$	$\{d \in \Delta^{\mathcal{J}} \mid \exists e \in C^{\mathcal{I}_i} : (d,e) \in r^{\mathcal{I}_i}\}$
Individuals:		
Named Individual	a	$a^{\mathcal{J}} \in \Delta^{\mathcal{J}}$

applying the rules from Figure 2 to derive new CIs, RIs and facts from an ontology $\mathcal{O}$, thus iteratively completing it. However, it is impossible to derive all subsumptions and facts, as both $\mathfrak{D}$ and $\mathbb{Z}$ (representing all time points) are infinite. Furthermore, two facts may also be separated by an arbitrarily large amount of time points depending on the chosen scale. If the algorithm cannot differentiate between relevant and irrelevant time points, it would need to access every time point, therefore increasing its runtime and memory usage. This might even make the algorithm infeasible for practical use.

Notice, on the other hand, that the only two diamond operators that can be the result of the operations $\cap$ and $\circ$ without appearing in their input are $\diamondsuit$ and $\boxplus$. Namely, $\diamondsuit \cap \diamondsuit = \diamondsuit$ and $\diamondsuit \circ \diamondsuit = \boxplus$. Therefore, the infinite number of diamond operators can be bypassed by restricting the rule application to the operators that appear in the ontology, in addition to $\boxplus$ and $\diamondsuit$. Similarly, the infinite amount of time points must be restricted so that the procedure can be applied to $\mathcal{O}$. First, consider $\mathsf{tem}(\mathcal{A})$, the set of all time points i appearing in $\mathcal{A}$. Next, consider the intervals between neighbouring elements from $\mathsf{tem}(\mathcal{A})$ and the intervals $(-\infty, \min(\mathsf{tem}(\mathcal{A})) - 1]$ and $[\max(\mathsf{tem}(\mathcal{A})) + 1, \infty)$. Now, an arbitrary number k can be chosen from such an interval $[i, j]$ to represent each number within the interval, denoted as $|l| := k$ for all $l \in [i, j]$. The set of all representative time points is constructed as follows.

$$\mathsf{rep}(\mathcal{A}) := \{|i| \mid i \in \mathbb{Z} \setminus \mathsf{tem}(\mathcal{A})\} \cup \mathsf{tem}(\mathcal{A})$$

By restricting all assertions to this finite set $\mathsf{rep}(\mathcal{A})$, the infiniteness of $\mathbb{Z}$ and the arbitrarily large gaps between entries in the ABox are dealt with.

As in Borgwardt et al. [5], for a cleaner representation, $\top$ and $\bot$ will be treated like concept names, thereby allowing assertions of the form $\top(a, i)$. For all concepts $A \in N_C$, roles $r \in N_R$ and Individuals $a, b \in N_I$, the abbreviations:

$$A(a) := \{i \in \mathsf{rep}(\mathcal{A}) \mid A(a, i) \in \mathcal{O}\}$$
$$r(a, b) := \{i \in \mathsf{rep}(\mathcal{A}) \mid r(a, b, i) \in \mathcal{O}\}$$

$$\text{T1}\ \frac{}{\diamondsuit_1 A \sqsubseteq A} \qquad \text{T2}\ \frac{}{\boxplus A \sqsubseteq \top} \qquad \text{T3}\ \frac{}{\diamondsuit_1 r \sqsubseteq r} \qquad \text{T4}\ \frac{\diamondsuit A_1 \sqsubseteq A_2 \quad \diamondsuit A_2 \sqsubseteq A_3}{(\diamondsuit \circ \diamondsuit) A_1 \sqsubseteq A_2}$$

$$\text{T5}\ \frac{\diamondsuit r_1 \sqsubseteq r_2 \quad \diamondsuit r_2 \sqsubseteq r_3}{(\diamondsuit \circ \diamondsuit) r_1 \sqsubseteq r_3} \qquad \text{T6}\ \frac{\diamondsuit A \sqsubseteq A_1 \quad \diamondsuit A \sqsubseteq A_2 \quad A_1 \sqcap A_2 \sqsubseteq B}{(\diamondsuit \cap \diamondsuit) A \sqsubseteq B}$$

$$\text{T7}\ \frac{}{\exists r.\bot \sqsubseteq \bot} \qquad \text{T8}\ \frac{\diamondsuit A \sqsubseteq \exists r.A_1 \quad \diamondsuit r \sqsubseteq s \quad \diamondsuit A_1 \sqsubseteq B_1 \quad \exists s.B_1 \sqsubseteq B}{\diamondsuit A \sqsubseteq B}$$

$$\text{T8}'\ \frac{\diamondsuit A \sqsubseteq \exists r.A_1 \quad \diamondsuit r \sqsubseteq s \quad \diamondsuit A_1 \sqsubseteq B_1 \quad \exists s.B_1 \sqsubseteq B \quad (\diamondsuit \cap \diamondsuit) \in \mathfrak{D}^{\pm}}{((\diamondsuit \cap \diamondsuit) \circ \diamondsuit) A \sqsubseteq B}$$

$$\text{A1}\ \frac{}{\top(a, i)} \qquad \text{A2}\ \frac{i \in \diamondsuit A(a) \quad \diamondsuit A \sqsubseteq B}{B(a, i)} \qquad \text{A3}\ \frac{i \in \diamondsuit r(a, b) \quad \diamondsuit r \sqsubseteq s}{s(a, b, i)}$$

$$\text{A4}\ \frac{A_1(a, i) \quad A_2(a, i) \quad A_1 \sqcap A_2 \sqsubseteq B}{B(a, i)} \qquad \text{A5}\ \frac{r(a, b, i) \quad A(b, i) \quad \exists r.A \sqsubseteq B}{B(a, i)}$$

Figure 2: Completion rules for $\mathcal{TELH}_{\bot}^{\diamondsuit,\mathsf{lhs}}$ ontologies [5]

are defined. Thus, $A(a)$ denotes the set of all time points at which a is in A, and $\Diamond\!\!\!\!\!+\!\!\!\!\!+A(a)$ refers to the set of time points at which a is inferred to satisfy $\Diamond\!\!\!\!\!+\!\!\!\!\!+A$ (given the assertions in $\mathcal{A}$, analogous to N from Section 2.2).

A, A_1, A_2, A_3, B, B_1 in the rules from Figure 2 are allowed to be instantiated by (*normalised*) $\mathcal{ELH}_\bot$ concepts, $\top$ or $\bot$ from $\mathcal{O}$, r, r_1, r_2, r_3, s by role names from $\mathcal{O}$, $\Diamond\!\!\!\!\!+\!\!\!\!\!+, \Diamond\!\!\!\!\!\bullet\!\!\!\!\!\bullet, \Diamond\!\!\!\!\!+$ by $\Diamond\!\!\!\!\!+\!\!\!\!\!+, \Diamond\!\!\!\!\!\bullet\!\!\!\!\!\bullet$ or elements of $\mathfrak{D}$ *occurring* in $\mathcal{O}$, a, b by individual names from $\mathcal{O}$, and i by values from $\mathrm{rep}(\mathcal{A})$, such that the resulting axioms are in normal form [5].

The rules in Figure 2 consist of two parts. The preconditions of a rule (above the horizontal line) specify what needs to be satisfied so that the conclusion (below the horizontal line) can be added to the ontology $\mathcal{O}$ if the concluded fact or assertion is not already in $\mathcal{O}$.

Also, note that the rules having CIs or RIs as a conclusion might produce redundancies because of the order of the diamond operators (Lemma 2). Consider the following example:

Example 1. The onology $\mathcal{O}$ consists of the following axioms:

(ax1): $\Diamond\!\!\!\!\!+\!\!\!\!\!+_1 A \sqsubseteq B$ (ax2): $\Diamond\!\!\!\!\!+\!\!\!\!\!+_{15} B \sqsubseteq C$ (ax3): $\Diamond\!\!\!\!\!+\!\!\!\!\!+ B \sqsubseteq D_1$ (ax4): $\Diamond\!\!\!\!\!+\!\!\!\!\!+ B \sqsubseteq D_2$ (ax5): $D_1 \sqcap D_2 \sqsubseteq C$.

Using the rules from Figure 2, the following derivations can be made:

$$\Diamond\!\!\!\!\!+\!\!\!\!\!+_{15} A \sqsubseteq C \quad \text{by T4 on (ax1) and (ax2)} \tag{2.1}$$

$$\Diamond\!\!\!\!\!+\!\!\!\!\!+ B \sqsubseteq C \quad \text{by T6 on (ax3), (ax4) and (ax5)} \tag{2.2}$$

$$\Diamond\!\!\!\!\!+\!\!\!\!\!+ A \sqsubseteq C \quad \text{by T4 on (ax1) and (2.2)} \tag{2.3}$$

These derivations make (ax2) and the CI $\Diamond\!\!\!\!\!+\!\!\!\!\!+_{15} A \sqsubseteq C$ redundant, as in both cases $\Diamond\!\!\!\!\!+\!\!\!\!\!+$ is the the stronger operator. Consequently, $\Diamond\!\!\!\!\!+\!\!\!\!\!+ B \sqsubseteq C$ already implies $\Diamond\!\!\!\!\!+\!\!\!\!\!+_{15} B \sqsubseteq C$ and $\Diamond\!\!\!\!\!+\!\!\!\!\!+ A \sqsubseteq C$ already implies $\Diamond\!\!\!\!\!+\!\!\!\!\!+_{15} A \sqsubseteq C$.

Thus, to circumvent these redundancies, a derived CI $\Diamond\!\!\!\!\!+ A \sqsubseteq B$ is only added to $\mathcal{O}$ if there is no other CI $\Diamond\!\!\!\!\!+ A \sqsubseteq B$ in $\mathcal{O}$ or, otherwise, if there already exists such a CI in $\mathcal{O}$, then the existing CI is modified by replacing the diamond operator with $(\Diamond\!\!\!\!\!+ \circ \Diamond\!\!\!\!\!+)$. Note, however, that this does not necessarily have to change the existing CI. Hence, $\mathcal{O}$ always includes only the unique strongest axiom of the form $\Diamond\!\!\!\!\!+ A \sqsubseteq B$ for any two concepts, A and B. RIs are handled in the same way.

The T(Box) rules implement the semantics of base cases and constructors in the context of CIs and RIs. Intuitively, T1 and T3 encode the meaning of the $\Diamond\!\!\!\!\!+\!\!\!\!\!+_1$-operator and, together with the selective addition or modification of CIs/RIs, "lay the foundation" for the semantics of diamond operators. T2 derives the trivial subsumptions that $\top$ must subsume every concept. T4 and T5 encode the semantics of subsumptions. The diamond operator of the conclusion, being the composition of the diamond operators from the preconditions, is also quite intuitive if one again imagines the application of diamond operators to concepts as "stretching occurrences of elements in the concept along the time axis." First, $\Diamond\!\!\!\!\!+$ "stretches" the occurrences of elements in A_1, and then $\Diamond\!\!\!\!\!\bullet\!\!\!\!\!\bullet$ "stretches" them again if it is a stronger operator. The meaning of conjunctions is encoded in rule T6, and it also makes intuitive sense that the weaker of the two input diamond operators is applied to the conclusion. T8 encodes the semantics of existential restrictions. T8' is a special case of T8, which might produce stronger axioms. This happens when $\Diamond\!\!\!\!\!\bullet\!\!\!\!\!\bullet$ and $\Diamond\!\!\!\!\!+$ are both non-convex operators and $\Diamond\!\!\!\!\!\bullet\!\!\!\!\!\bullet r(a,b) \cap \Diamond\!\!\!\!\!+ A_1(b)$, for any two elements a, b, must either be empty or an interval where at least one of the boundaries is ∞ or $-\infty$. Thus, the temporal operator of the resulting axiom must be non-convex as well. T7, together with T8 and T8' propagates subsumptions by $\bot$ in existential restrictions. Thus, enforcing that concepts which are subsumed by existential restrictions involving an empty concept must themselves be empty.

The A(Box) rules apply the $\mathcal{TELH}_\bot^{\Diamond\!\!\!\!\!+\!\!\!\!\!+,\mathsf{lhs}}$ semantics to assertions with concrete named individuals. First, A1 derives the trivial fact that all individuals are elements of $\top$ at all times. A2 and A3 implement the meaning of subsumptions. A4 and A5 encode the semantics of the constructors conjunction and existential restriction, respectively.

Example 2. Consider the ontology $\mathcal{O}$ describing observational weather data. The elements in the domain are cities. The precipitation is measured in 10-minute intervals, and moments with rain are registered in the concept Rain. The windspeed is measured every minute, and a location gets inserted into the concept $\mathit{HighWindspeed}$ if the windspeed exceeds 75 km/h. The TBox $\mathcal{T}$ of $\mathcal{O}$ consists of the following axioms.

$$\text{(ax1): } \Diamond_{11}\mathit{Rain} \sqsubseteq \mathit{Rain} \quad \text{(ax2): } \Diamond_{20}\mathit{HighWindspeed} \sqsubseteq \mathit{Storm}$$
$$\text{(ax3): } \Diamond_{1}\mathit{WeatherPhenomenon} \sqsubseteq \mathit{Rain} \quad \text{(ax4): } \Diamond_{1}\mathit{WeatherPhenomenon} \sqsubseteq \mathit{HighWindspeed}$$
$$\text{(ax5): } \mathit{Rain} \sqcap \mathit{Storm} \sqsubseteq \mathit{Thunderstorm}.$$

It is assumed that if rain fell during two consecutive measurements, it was raining as well in between. Additionally, it is assumed that if the windspeed exceeded 75 km/h twice within 20 minutes, there was a storm above the city. The following axioms can be derived, although the list is not exhaustive.

$$\Diamond_{1}\mathit{Storm} \sqsubseteq \mathit{Storm} \quad \text{by T1} \tag{2.4}$$
$$\Diamond\!\!\!\!-\mathit{Thunderstorm} \sqsubseteq \top \quad \text{by T2} \tag{2.5}$$
$$\Diamond_{20}\mathit{WeatherPhenomenon} \sqsubseteq \mathit{Storm} \quad \text{by T4 on (ax2) and (ax4)} \tag{2.6}$$
$$\Diamond_{1}\mathit{WeatherPhenomenon} \sqsubseteq \mathit{Thunderstorm} \quad \text{by T6 on (ax3), (ax5) and (2.6)} \tag{2.7}$$

The ABox $\mathcal{A}$ of $\mathcal{O}$ consists of the following assertions.

$$\text{(ax6): } \mathit{Rain}(\text{Berlin}, 1) \quad \text{(ax7): } \mathit{Rain}(\text{Berlin}, 6)$$
$$\text{(ax8): } \mathit{HighWindspeed}(\text{Berlin}, 4) \quad \text{(ax9): } \mathit{HighWindspeed}(\text{Berlin}, 21).$$

$\mathrm{tem}(\mathcal{A})$ includes the elements $1, 4, 6$, and 21. Thus, $\mathrm{rep}(\mathcal{A}) = \{-1, 1, 2, 4, 5, 6, 7, 21, 34\}$ where $-1, 2, 5, 7$, and 34 are representatives of the intervals between the elements from $\mathrm{tem}(\mathcal{A})$. The following facts can be derived.

$$\mathit{Rain}(\text{Berlin}, 2) \quad \text{by A2 on (ax1), (ax6), and (ax7)} \tag{2.8}$$
$$\mathit{Rain}(\text{Berlin}, 4) \quad \text{by A2 on (ax1), (ax6), and (ax7)} \tag{2.9}$$
$$\mathit{Rain}(\text{Berlin}, 5) \quad \text{by A2 on (ax1), (ax6), and (ax7)} \tag{2.10}$$
$$\mathit{Storm}(\text{Berlin}, 4) \quad \text{by A2 on (ax2), (ax8), and (ax9)} \tag{2.11}$$
$$\mathit{Thunderstorm}(\text{Berlin}, 4) \quad \text{by A4 on (ax5), (2.9), and (2.11)} \tag{2.12}$$

This example illustrates how the algorthim deals with temporal gaps in the ABox.

2.4 The Web Ontology Language

Tools like Protégé [18] already exist that can be used to construct ontologies in *OWL*. These tools are convenient, and this standard is a well-established method of specifying ontologies in a machine-readable format. Therefore, the algorithm from Borgwardt, Forkel, and Kovtunova [5] will be implemented to use ontologies as input, which are specified in a sublanguage of OWL 2 corresponding to $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$. OWL is an extension of the *RDF*. Thus, this section provides a brief overview of RDF, some of its extensions, and its relation to DLs.

The *World Wide Web Consortium* (*W3C*) published a first *RDF* specification in 1999. Although the framework was initially intended to represent metadata of web resources, it has evolved over the years into a more general formal language for describing structured information, making the semantics of a resource machine-readable [11, Ch. 2]. The main idea is to represent the information in the form of subject-predicate-object statements, where the subject and object are resources or simple values, and the predicate describes the relationship between them.

Table 2: The correspondences between OWL 2 EL and $\mathcal{ELH}_\bot$ [11]

	$\mathcal{ELH}_\bot$	OWL 2 EL
Top	$\top$	`owl:Thing`
Bottom	$\bot$	`owl:Nothing`
Concept	A	`(A, rdf:type, owl:Class)`
Role	r	`(r, rdf:type, owl:ObjectProperty)`
Individual	a	An individual is just a resource, which may be attributed to a class: `(a, rdf:type, A)`
Concept Inclusion	$C \sqsubseteq D$	`(C, rdfs:subClassOf, D)`
Role Inclusion	$r \sqsubseteq s$	`(r, rdfs:subPropertyOf, s)`
Axiom (in general)	α e.g. $C \sqsubseteq D$	The axiom α is represented by a *blank node* `_:1`: `(_:1, rdf:type, owl:Axiom)`, `(_:1, owl:annotatedSource, C)`, `(_:1, owl:annotatedProperty, rdfs:subClassOf)`, `(_:1, owl:annotatedTarget, D)`
Existential Restriction	$\exists r.C$	`(_:1, rdf:type, owl:Restriction)`, `(_:1, owl:onProperty, r)`, `(_:1, owl:someValuesFrom, C)`
Conjunction	$C \sqcap D \sqcap \ldots$	`(_:1, owl:intersectionOf, _:2)`, `(_:2, rdf:first, C)`, `(_:2, rdf:rest, _:3)`, `(_:3, rdf:first, D)`, . . . Here the blank node `_:1` represents the conjunction of the classes within the linked list starting at `_:2`

These tuples (subject, predicate, object) are also called *triples* and comprise an RDF document. Such a document thus describes a directed, labelled graph, where the nodes represent resources or values, and the edges represent the relationships [11, Sec. 2.2]. *Internationalized Resource Identifiers* (*IRIs*) are used to unambiguously label resources and relationships [11, 6]. A node can also refer to a concrete data value like a string or a number. In that case, the node represents a so-called *literal* and is labelled with the specific value [11, Sec. 2.1]. A third special kind of node is a *blank node*, which is used to represent resources without a specific IRI. They are used as distinct but anonymous placeholders when modelling many-valued relationships [11, Sec. 2.3.4].

The *RDF Schema* (*RDFS*) extends RDF by providing a vocabulary for describing classes and properties, where classes correspond to concepts and properties to roles. Thus, RDFS enables the creation of ontologies. Therefore, it is called an *ontology language* [11, Sec. 2.4]. As RDFS has its limitations, *OWL* was developed to create more expressive ontologies [11, Ch. 4]. The current version is *OWL 2*. *OWL 2 EL* is a tractable sublanguage of OWL 2, which corresponds to, as its name suggests, the DL $\mathcal{EL}^{++}$ (an extension of $\mathcal{ELH}_\bot$ [4]) and is used as the base language before "temporalising" ontologies. Table 2 shows how syntactic elements from $\mathcal{ELH}_\bot$ are constructed as triples in OWL 2 EL. The prefixes `rdf:`, `rdfs:` and `owl:` are used as abbreviations for the namespaces `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, `http://www.w3.org/2000/01/rdf-schema#` and `http://www.w3.org/2002/07/owl#`, respectively [11].

It is important to note that `owl:Axiom`s are used to encapsulate more complex axioms. This is unnecessary for simple CIs, as seen in Table 2. However, OWL permits the addition of annotation properties like comments or labels to axioms. To satisfy this requirement, the simple CI is represented by a blank node `_:1` and a new

triple `(_:1, rdfs:comment, "This is a generic comment")` is added to the ontology. The same can be done for RIs. Additionally, complex concepts like existential restrictions are represented by their blank node within CIs. For example, the CI $\exists r.C \sqsubseteq D$ might be encoded as `(_:1, rdf:type, owl:Restriction)`, `(_:1, owl:onProperty, r)`, `(_:1, owl:someValuesFrom, C)`, `(_:1, rdfs:subClassOf, D)`.

2.5 The *Nemo* Rule Engine

Nemo is one of the most recent additions to the vast field of Datalog-based rule engines [12, 14]. As such, this section provides a brief and informal overview of pure Datalog before diving into *Nemo's* more specific Datalog dialect and providing an example of an implementation of a reasoning algorithm in *Nemo*.

Datalog is a declarative database query language which essentially adapts the logic programming paradigm to relational databases [1, 7]. Similarly to languages like Prolog, Datalog programs consist of a finite set of "if-then" *rules* of the form:

$$L_0 \leftarrow L_1, \ldots, L_n.$$

The rule can be read as "if L_1 and $\ldots$ and L_n, then L_0." Each L_i is an *atom* $p_i(t_1, \ldots, t_{k_i})$, and t_i is a term. Terms may be *constants* or *variables*. An essential feature of Datalog is that it permits recursion. The left-hand side of $\leftarrow$ is called the rule's *head*, and the right-hand side is called the rule's *body*, which may be empty. If it is, the rule is called a *fact*. Thereby, facts denote (analogous to an ABox) assertions of concrete information, while deductive rules represent relations (analogous to TBox axioms), which make it possible to deduce facts from other facts [7]. An example program looks like the following:

$$\begin{aligned} parentOf(alice, bob) &\leftarrow \\ parentOf(bob, dee) &\leftarrow \\ grandparentOf(Z, X) &\leftarrow parentOf(Z, Y), parentOf(Y, X) \end{aligned}$$

where $alice$, bob and dee are constants, and X, Y and Z are variables. $parentOf(alice, bob)$ and $parentOf(bob, dee)$ are this example program's facts, which denote that $alice$ is a parent of bob and bob, in turn, is a parent of dee. Using the rule

$$grandparentOf(Z, X) \leftarrow parentOf(Z, Y), parentOf(Y, X),$$

the fact $grandparentOf(alice, dee)$ can be computed.

There exist several different but equivalent approaches to defining the semantics of Datalog programs. The first approach is *model theoretic*, where the rules are viewed as logical sentences that specify which properties the result must satisfy. The unique result is then the smallest set of facts, which makes the sentences true. The second approach is *proof-theoretic*, where facts are in the result if a proof of them can be obtained from the rules. The *fixpoint* approach defines the semantics of a program as a particular solution of a fixpoint equation [1].

Although Datalog and logic programming are closely related, some key distinctions exist. First, differing underlying assumptions are made about how the data is supplied for a program. A logic program is assumed to contain all the necessary facts and deductive rules within itself [7]. On the other hand, Datalog programs are assumed to run on a database, thus having a comparatively large number of facts and fewer rules. Second, logic programming allows function symbols, for example, to represent complex data structures such as lists, whereas Datalog does not permit them [1].

The decision problem for Datalog is as follows. Given a Datalog program split into a set of facts and a set of rules with non-empty bodies and a set of atoms without variables, do the atoms follow from the program? Keeping the set of facts fixed, whereas the set of rules and the set of atoms are inputs (the so-called *data complexity*), this problem is *P*-complete [8]. Using the rules and atoms as input while keeping the set of facts fixed (the so-called *program complexity*) results in Datalog being *ExpTime*-complete [8]. However, introducing arithmetic constraints [8] or tuple generation [15] (existential rules) makes query answering undecidable.

In *Nemo's* Datalog dialect, the program from above would look like this:

```
parentOf(alice, bob) .
parentOf(bob, dee) .
grandparentOf(?Z, ?X) :- parentOf(?Z, ?Y), parentOf(?Y, ?X) .
```

In this simple case, the syntax looks almost identical to pure Datalog. The few differences are that all rules end with `.`, variables are prepended with `?`, and `:-` replaces $\leftarrow$. In addition, *Nemo* increases its syntax's expressivity with several valuable features. The most crucial features used in the implementation in Chapter 3 are listed below.

Nemo's support for importing external data as predicates, especially ontologies specified in OWL 2, from files in formats such as *RDF/XML*, *Turtle*, *N-Triples*, and *CSV* is an essential feature. Of course, it also allows the export of predicates [12].

```
    @import triple :- rdf{resource="example.ttl"} .
    @export result :- csv{resource="result.csv"} .
```

Although *Nemo* does not allow the definition of one's own function symbols, it supplies a number of built-in functions. For example, entries can be cast from one datatype to another with functions like `INT(X)` or `STR(X)`. More complex string operations can be accomplished with `START(X, Y)`, specifying if `X` starts with `Y`, `SUBSTR(X, i)`, returning the substring of `X` starting at index `i`, and several more [12]. It is also possible to add filters to rules, making statements like the subsequent one possible.

```
primarySchoolPupil(?X) :- pupil(?X), schoolYear(?X, ?year), ?year < 5 .
```

This rule collects all primary school pupils from all students.

Additionally, *Nemo* supports the generation of new elements within so-called *existential rules*, which stipulate that a value exists but not what it exactly is. More specifically, this creates *blank nodes* (also *nulls* or *named nulls*), distinct but anonymous placeholders [12]. For example, the following program generates unique identifiers for students.

```
student(!Id, ?Name, ?DateOfBirth) :-
    personalInfo(?Name, ?DateOfBirth), enrolled(?Name) .
```

Furthermore, *Nemo* allows the use of negations within stratified programs. This means that atoms may be negated as long as they are not involved in recursive dependency cycles [12]. To negate a predicate, it is prefixed with `~`.

Similar to the database language SQL, *Nemo* supports aggregates, for example, to determine the tuple containing the maximal value within a predicate [12]. For example:

```
pupil("cee") .
pupil("dee") .
age("cee", 7) .
age("dee", 6) .
eldestPupil(?X, #max(?age)) :- pupil(?X), age(?X, ?age) .
```

After the program's execution, the predicate `eldestPupil` contains the tuple `("cee", 7)`.

Crucially, *Nemo* does not support the modification or removal of tuples from any predicates. This must be taken into consideration in the $\mathcal{TELH}_{\perp}^{\diamondsuit,\mathsf{lhs}}$ completion algorithm implementation because the algorithm originally envisages the modification of diamonds added to CIs and RIs, as discussed in Section 2.3.

2.5.1 An Example Reasoner in *Nemo*

The Knowledge-Based Systems research group at TU Dresden, which developed *Nemo*, provides an online demonstration[1] of implementing the rules from the ELK reasoner. These rules constitute a completion algorithm for ontologies in $\mathcal{EL}_{\perp}^{+}$ [13], which is equivalent to $\mathcal{ELH}_{\perp}$, except that it also allows the composition of roles. These rules are similar to those in Figure 2, but as the input ontologies are specified in OWL 2 EL triples (see Section 2.4), they are not directly applicable. The nested structures which encode DL constructors like conjunctions (see Table 2), thus, need to be resolved before the rules become applicable. The example covers the necessary preprocessing steps to normalise the input ontology. As the implementation in Chapter 3 faces the same issue, but in a less pronounced way, some of these steps/ideas are carried over, which is why the example is included here.

The preprocessing starts by collecting all classes which appear in the input.

```
ClassObject(owl:someValuesFrom) .
ClassObject(rdf:first) .
ClassObject(rdfs:subClassOf) .
ClassObject(owl:equivalentClass) .
ClassSubject(rdfs:subClassOf) .
ClassSubject(owl:equivalentClass) .
class(?O) :- TRIPLE(?X, ?P, ?O), ClassObject(?P) .
class(?X) :- TRIPLE(?X, ?P, ?O), ClassSubject(?P) .
```

Here, the predicates `ClassObject` and `ClassSubject` are defined. They contain the information for which OWL/RDFS/RDF properties a class appears as the object or as the subject, respectively. The predicate `class` then collects all appearing classes. Thereby, it does not differentiate between classes labelled with IRIs and blank nodes representing complex classes.

The next step is to mark blank nodes as auxiliary classes. Consider existential restrictions, for example.

```
synEx(?Y,?P,?X), auxClass(?X) :-
    TRIPLE(?X, owl:someValuesFrom, ?Y), TRIPLE(?X, owl:onProperty, ?P) .
```

This rule accumulates all existential restrictions in the predicate `synEx` and marks `?X` as an auxiliary class, as the blank node represents the restriction. With all the auxiliary classes marked as such, the classes "named" with IRIs can be collected. They are equivalent to atomic concepts.

```
nf:isMainClass(?X) :- class(?X), ~auxClass(?X) .
```

[1] https://tools.iccl.inf.tu-dresden.de/nemo/#

Now, the existential restrictions `synEx(?Y,?P,?X)` can recursively be normalised. This is necessary because the class denoted by `?Y` might be an auxiliary class itself. The following rules are used for this purpose.

```
1  repOf(?X,?X) :- nf:isMainClass(?X) . % keep main classes unchanged
2  synExRep(?X,?P,?Rep) :- synEx(?Y,?P,?X), repOf(?Y,?Rep) .
3  nf:exists(!New,?P,?Rep) :- synExRep(?X,?P,?Rep) .
4  repOf(?X,?N) :- synExRep(?X,?P,?Rep), nf:exists(?N,?P,?Rep) .
```

Line 1 represents the "base case" of the `repOf` predicate, thereby stating that each "atomic" class represents itself. Then, in line 2, `synExRep` collects all existential restrictions and replaces `?Y` with the class it represents. In line 3, all the normalised existential restrictions are collected. `nf:` is an abbreviation for the namespace `http://rulewerk.semantic-web.org/normalForm/`. The use of `!New` guarantees that each normalised existential restriction is represented by a unique identifier. The use of an existential rule is not a problem for the termination of this program, as *Nemo* only creates a new tuple per normalised existential restriction. Finally, line 4 contains the "recursive case" where the previous blank node is registered as the representative of the newly created one. Note that the preprocessing of conjunctions follows the same principles; thus, `repOf` also contains blank nodes and the conjunctions they represent.

The marking of classes and the division into "atomic" and auxiliary classes are carried over to the implementation in Chapter 3. Otherwise, the preprocessing can be simplified. The preprocessing in the ELK example also contains some more steps, but they are less relevant for the later implementation.

Next, two examples should suffice to show how the rules specified in the ELK reasoner can be translated into *Nemo* syntax. First the rule [13]

$$\mathsf{R}_0 \ \frac{\text{init}(C)}{C \sqsubseteq C}$$

is translated into *Nemo* syntax as

```
1  inf:subClassOf(?C,?C) :- init(?C) .
```

where `inf:` is an abbreviation for the namespace `http://rulewerk.semantic-web.org/inference/`. Thus, `inf:subClassOf` should contain the inferred subsumptions. The second example is the more complex rule [13]

$$\mathsf{R}^+_\sqcap \ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occurs negatively in } \mathcal{O}.$$

Its equivalent in *Nemo* syntax is the following:

```
1  inf:subClassOf(?C,?Y) :-
2      inf:subClassOf(?C,?D1), inf:subClassOf(?C,?D2),
3      nf:conj(?Y,?D1,?D2), nf:isSubClass(?Y) .
```

A (complex) concept C negatively appearing in an ontology means an axiom $D \sqsubseteq E$ exists, where C is a syntactic subexpression of D [13]. The exact meaning of init(C) and the concrete ideas behind each of these rules are not relevant. Nevertheless, it should be evident that once all the necessary predicates are created, the translation of the rules is relatively straightforward.

3 An Implementation Of The Completion Algorithm

This chapter discusses the implementation of the completion algorithm for $\mathcal{TELH}_{\bot}^{\Diamond,\mathsf{lhs}}$ from [5] in the *Nemo* rule engine. In this implementation, the whole reasoning process is divided into two parts. First is the computation of all possible entailments within the TBox of the provided ontology. Second, the program uses the completed TBox to deduce all possible facts from the ABox. This can be done, as the TBox reasoning is entirely independent of the result from the ABox reasoning, and the ABox's exhaustive completion depends on the completed TBox. Each part also requires some preprocessing in order to enter the provided data correctly into predicates.

3.1 The Temporalisation of OWL

Section 2.4 described the general syntax of the OWL 2 EL standard, which originally corresponds to the atemporal DL $\mathcal{EL}^{++}$ [6]. To encode the subsumption $\Diamond C \sqsubseteq D$, one would, in theory, need to add a fourth entry to the triple `(C, rdfs:subClassOf, D)`. However, the name *triple* already makes clear that this would break the standard. Thus, any subsumption including a temporal concept cannot be directly added as a triple into an OWL ontology. Instead, the subsumption is encapsulated in an `owl:Axiom` (see Table 2) to keep the encoding standard-compliant. The same applies to individuals which have time points attached to their occurrences in classes. OWL 2 EL allows the addition of user-defined annotation properties to axioms. They get attached as another triple `(_:1, exampleProperty, "This is a new annotation Property")`, where `_:1` is the blank node representing the `owl:Axiom`. Note that the standard also permits the attachment of multiple annotation properties to one axiom. Thus, ABox assertions where the same individual satisfies the same concept at several points in time can be encoded as well. The structure of a temporal axiom $\Diamond C \sqsubseteq D$ is visualised in Figure 3.

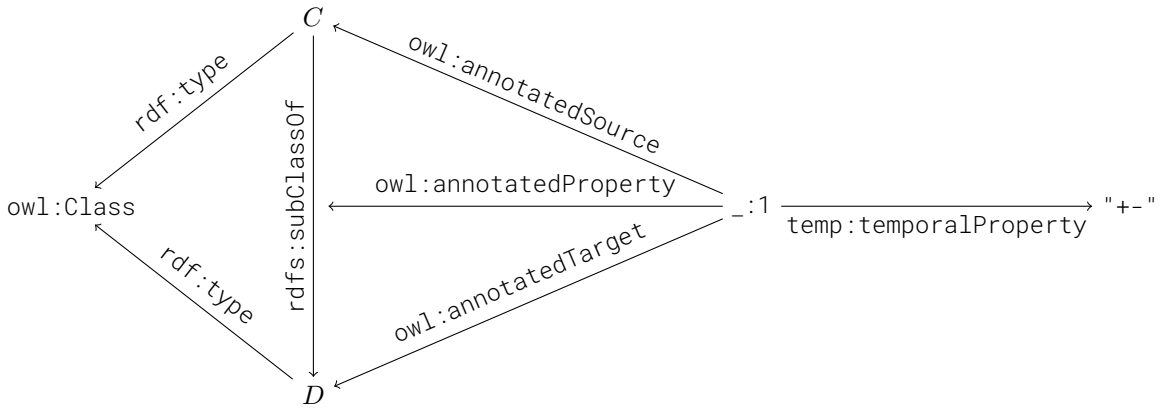


Figure 3: The structure of a temporal axiom $\diamondsuit C \sqsubseteq D$ in an OWL-compliant format

The chosen name for the annotation property is `temp:temporalProperty`. In an overloading manner, this property is used to encode the temporal operators as well as the time points of occurrences of individuals in classes. The value of this property is a string representing the diamond operator if it is attached to a CI/RI or an integer if it is attached to an `owl:Axiom` encapsulating the occurrences of an individual in a class.

Notice that the operator $\diamondsuit$ is encoded as "+-" in Figure 3. The remaining operators are encoded in a similar fashion, as can be seen in Table 3. The naming is chosen arbitrarily, but this is an easy, unambiguous, and concise way to do it.

This simple encoding is only possible as the TBox is required to be normalised. For example, to encode $(\diamondsuit A) \sqcap (\diamondsuit B) \sqcap (\exists(\diamondsuit r).C) \sqsubseteq D$, the temporal operators would need to be attached to the individual concepts/roles within the axiom, instead of the axiom as a whole.

3.2 Reading the TBox in

This section continues where Section 2.5.1 left off by discussing the preprocessing steps necessary to read the TBox in from an OWL ontology. As mentioned there, the code snippet marking all classes is carried over.

```
1  ClassObject(owl:someValuesFrom) .
2  ClassObject(rdfs:subClassOf) .
3  ClassObject(owl:equivalentClass) .
4  ClassSubject(rdfs:subClassOf) .
5  ClassSubject(owl:equivalentClass) .
6  class(?O) :- triples(?X, ?P, ?O), ClassObject(?P) .
7  class(?X) :- triples(?X, ?P, ?O), ClassSubject(?P) .
8
9  property(?R) :- triples(?R, rdf:type, owl:ObjectProperty) .
```

Table 3: The diamond operators and their corresponding comments

Diamond Operator	Comment
$\diamondsuit$	"+-"
$\diamondsuit_+$	"+"
$\diamondsuit_-$	"-"
$\diamondsuit_c$	"con"
$\diamondsuit_n$	"c*n*", e.g. $\diamondsuit_4$ is encoded as "c4"

The only addition here is that properties are also marked as such. The rest of the preprocessing of conjunctions and existential restrictions can be simplified as this algorithm expects already normalised ontologies as input. For example, conjunctions can be marked and collected directly, instead of first having to completely deconstruct the nested list representing the conjunction (see Table 2).

```
conj(?idConj, ?C1, ?C2), auxClass(?idConj) :-
    triples(?idConj, owl:intersectionOf, ?idFirst), triples(?idFirst, rdf:first, ?C1),
    triples(?idFirst, rdf:rest, ?idRest), triples(?idRest, rdf:first, ?C2) .
```

Notice that this rule also accumulates the blank nodes representing auxiliary classes in the predicate `auxClass`. The same goes for existential restrictions in the predicate `existRestrict`. Now, "main" classes are marked the same way as seen in Section 2.5.1.

```
nf:isMainClass(?X) :- class(?X), ~auxClass(?X) .
```

From here on out, the preprocessing differs from the ELK example, as now axioms need to be deserialised, and their attached annotation properties need to be taken into account, which previously could safely be ignored. The first step is to collect all temporal axioms.

```
tempAxiom(?tempOp, ?C, rdfs:subClassOf, ?D), appearingTempOp(?tempOp) :-
    triples(?id, rdf:type, owl:Axiom),
    triples(?id, owl:annotatedSource, ?C),
    triples(?id, owl:annotatedTarget, ?D),
    triples(?id, owl:annotatedProperty, rdfs:subClassOf),
    triples(?id, temp:temporalProperty, ?tempOp) .
```

The conditions of this rule directly follow from the structure shown in Figure 3. The predicate `commentType` is used to filter out the triples which contain comments/labels. Its only two entries are the RDFS properties `rdfs:comment` and `rdfs:label`. Notice also the predicate `appearingTempOp` collecting the temporal operators appearing in the input. Additionally, the atemporal axioms are collected as well.

```
nonTempAxiom(?C, rdfs:subClassOf, ?D) :-
    triples(?C, rdfs:subClassOf, ?D), ~tempAxiom(_, ?C, rdfs:subClassOf, ?D) .
```

Now, all the collected axioms can be combined into one 4-ary predicate `axiom`. The first argument is the temporal operator, the second and fourth are the connected classes, and the third is the connecting property. Temporal axioms can be added as they are.

```
axiom(?tempOp, ?C, ?type, ?D) :- tempAxiom(?tempOp, ?C, ?type, ?D) .
```

Axioms with conjunctions or existential restrictions need to be considered differently, as the normal form mentioned in Section 2.3 does not permit them to be labelled with a diamond.

```
impossibleTempOp(?id, ?C, ?D) :- conj(?id, ?C, ?D) .
impossibleTempOp(?id, ?R, ?D) :- existRestrict(?id, ?R, ?D) .

axiom("not allowed", ?C, ?type, ?D) :-
    nonTempAxiom(?C, ?type, ?D), impossibleTempOp(?C, _, _) .
```

Instead of having a temporal operator as their first entry, they receive the placeholder string "`not allowed`". Other atemporal axioms, which are subsumptions of two $\mathcal{ELH}_\bot$ concepts, are added with "`c1`" as their diamond, as described in Section 2.3.

```
axiom("c1", ?C, ?type, ?D) :-
    nonTempAxiom(?C, ?type, ?D), ~impossibleTempOp(?C, _, _) .
```

Although the rules shown here only cover CIs, the same processing is applied to RIs.

The second major part of the TBox's encoding in predicates is the implementation of the order on the diamonds from Lemma 2, as *Nemo* does not directly support list datatypes. First, the diamond operators, which might not appear in the input ontology but could be added during the reasoning process (see Section 2.3), are added to the predicate `appearingTempOp`:

```
appearingTempOp("c1") .
appearingTempOp("+-") .
appearingTempOp("+") .
appearingTempOp("-") .
appearingTempOp("con") .
```

Although $\boxplus$ and $\boxminus$ cannot be newly derived during the reasoning, they are incomparable and therefore, their order needs to be considered separately. Thus, for consistency's sake, they are added as well. Next, the metric convex diamonds are collected and stored together with their specified interval length in the predicate `metricTempOp`:

```
metricTempOp(?Op, INT(SUBSTR(?Op, 2))) :-
    appearingTempOp(?Op), isTrue(STRSTARTS(?Op, "c")),
    isTrue(isInteger(INT(SUBSTR(?Op, 2)))) .
```

Here, `isTrue` is an auxiliary predicate which is only satisfied by the boolean value "true". This is necessary because *Nemo* would otherwise consider the built-in function `isInteger` as a predicate. The "if"-condition of this rule can be read as follows. `?Op` needs to be an operator, which appears in the input ontology and the string, which encodes the diamond, must start with a "c". Additionally, the substring of `?Op` starting at index 2 must be a number. Checking this is possible, as the function `INT` returns nothing if the supplied string is not an integer.

Now, the order of the diamonds can be computed by collecting all tuples $(\boxplus, \boxdot)$, such that $\boxplus \subseteq \boxdot$ holds. First, the order on $\mathfrak{D}^{\pm} \cup \{\boxplus\}$ is added:

```
tempOpSequence("+", "+-") .
tempOpSequence("-", "+-") .
tempOpSequence("con", "+") .
tempOpSequence("con", "-") .
```

Next, the tuples expressing the reflexivity of the diamonds are added together with the relation from every metric convex diamond to $\boxplus$:

```
tempOpSequence(?op, ?op) :- appearingTempOp(?op) .
tempOpSequence(?op, "con") :- metricTempOp(?op, ?i) .
```

Now, the order on the metric convex diamonds is imposed by comparing their interval lengths:

```
tempOpSequence(?op1, ?op2) :-
    metricTempOp(?op1, ?i1), metricTempOp(?op2, ?i2), ?i1 < ?i2 .
```

To complete the order, the transitive closure is computed by adding all tuples $(\boxplus, \boxdot)$, such that there exists a diamond $\boxplus$ with $(\boxplus, \boxplus)$ and $(\boxplus, \boxdot)$ in the predicate `tempOpSequence`.

```
tempOpSequence(?op1, ?op2) :-
    tempOpSequence(?op1, ?opInterm), tempOpSequence(?opInterm, ?op2) .
```

Having imposed the order, the intersection and composition of diamond operators can be defined as 3-tuples, which have the result from the respective operation as their third element. The intersection is defined as follows:

```
1  tempOpIntersection("+", "-", "con") .
2  tempOpIntersection("-", "+", "con") .
3
4  tempOpIntersection(?op1, ?op2, ?op1), tempOpIntersection(?op2, ?op1, ?op1) :-
5      tempOpSequence(?op1, ?op2) .
```

The lines 1 and 2 add the special case $\Diamond_? \cap \Diamond_? = \Diamond_? \cap \Diamond_? = \Diamond_?$. Lines 4 and 5 add the general case, where the intersection of the two diamonds is the smaller entry in the corresponding tuple in `tempOpSequence`. The composition is defined in a similar way.

This finalises the encoding of the TBox. The next section will discuss the translation of the TBox rules into *Nemo* syntax.

3.3 Translation of the TBox rules

As seen in Section 2.5.1, the translations of the algorithm's rules are quite straightforward, as all the necessary predicates are already computed. Therefore, the translation of all the rules is omitted here, and instead, only two examples are discussed. The first example is the rule

$$\text{T1}\ \frac{}{\Diamond_1 A \sqsubseteq A}\ .$$

It translates to the following *Nemo* rule.

```
inf:axiom("c1", ?A, rdfs:subClassOf, ?A) :- nf:isMainClass(?A) .
```

The prefix `inf:` is used to indicate that the predicate `inf:axiom` contains the inferred axioms in addition to the ones from the input ontology. The condition `nf:isMainClass(?A)` enforces that this rule is only instantiated for "named" classes. The second example is the rule

$$\text{T8}\ \frac{\Diamond A \sqsubseteq \exists r.A_1 \quad \Diamond r \sqsubseteq s \quad \Diamond A_1 \sqsubseteq B_1 \quad \exists s.B_1 \sqsubseteq B}{\Diamond A \sqsubseteq B}\ .$$

Although the rule is more complex, the translation is still almost immediate.

```
inf:axiom(?tempOp1, ?A, rdfs:subClassOf, ?B) :-
    inf:axiom(?tempOp1, ?A, rdfs:subClassOf, ?IdExRest1),
    existRestrict(?IdExRest1, ?R, ?A1),
    inf:axiom(?tempOp2, ?R, rdfs:subPropertyOf, ?S),
    inf:axiom(?tempOp3, ?A1, rdfs:subClassOf, ?B1),
    inf:axiom("not allowed", ?IdExRest2, rdfs:subClassOf, ?B),
    existRestrict(?IdExRest2, ?S, ?B1),
    tempOpIntersection(?tempOp2, ?tempOp3, ?tempOpIntersection),
    ~tempOpPlusMinus(?tempOpIntersection) .
```

The extra conditions `existRestrict(?IdExRest1, ?R, ?A1)` and `existRestrict(?IdExRest2, ?S, ?B1)` verify that the necessary concepts are indeed existential restrictions. This rule includes a small optimisation, as it additionally specifies the

filter $(\diamondsuit \cap \boxplus) \notin \mathfrak{D}^{\pm}$. This ensures that the rules T8 and T8' are not applied to the same axioms which avoids redundant entailments. The predicate `tempOpPlusMinus` is equivalent to $\mathfrak{D}^{\pm}$. Thus, the conditions `tempOpIntersection(?tempOp2, ?tempOp3, ?tempOpIntersection)` and `~tempOpPlusMinus(?tempOpIntersection)` accomplish the filter $(\diamondsuit \cap \boxplus) \notin \mathfrak{D}^{\pm}$.

At this point, the TBox includes the aforementioned redundancies, as all the program did was add new axioms to the already inferred ones. Consider Example 1 and assume that the given axioms were part of a larger TBox. Now, suppose that through other derivations, the following two new CIs are added.

$$\diamondsuit_1 C \sqsubseteq E \tag{3.1}$$

$$\diamondsuit C \sqsubseteq E \tag{3.2}$$

This creates four different ways to apply T4 to CIs of the form $\diamondsuit A \sqsubseteq C$ and $\diamondsuit C \sqsubseteq D$ (eq. (2.1) and eq. (3.1), eq. (2.1) and eq. (3.2), eq. (2.3) and eq. (3.1), eq. (2.3) and eq. (3.2)). Each combination is checked and three new CIs are added, two of which are redundant. This behaviour cannot be circumvented during the TBox reasoning phase.

Either one accepts the redundancies and continues with the ABox reasoning phase, or one simulates the intended behaviour of the TBox rules and provides the ABox reasoning phase with a redundancy-free TBox. This can be achieved in several ways. One option would be to export the finished TBox and write a script in another language like Python which would eliminate the redundant axioms. A second option would be to declare a new predicate within the *Nemo* program and filter `inf:axiom` for the strongest CIs/RIs. The third alternative is to combine both approaches, thereby splitting the TBox and ABox reasoning phase into separate programs and using a third *Nemo* program to clear the TBox of redundancies. The last option is chosen here, as this simplifies an analysis of both the TBox and ABox reasoning phase.

The aforementioned *Nemo* program is quite simple. All it does is first classify all axioms by their associated temporal operator, thereby collecting all axioms with $\boxplus$ in one predicate, all axioms with $\diamondsuit$ in another and so on. Then, for each axiom $\diamondsuit A \sqsubseteq B$ it checks the predicates with stronger temporal operators first before adding the axiom to the adjusted TBox. The following representative rule shows the collection of all axioms with $\boxplus$.

```
plusMinusAxiom(?A, ?Connector, ?B) :- inf:axiom("+-", ?A, ?Connector, ?B) .
```

A special case are the metric temporal operators as they are first collected in a separate predicate and then the strongest metric temporal operator is determined through the use of the aggregate function `#max`.

```
metricAxiom(?Op, ?i, ?A, ?Connector, ?B) :-
    inf:axiom(?Op, ?A, ?Connector, ?B), metricTempOp(?Op, ?i) .
maxMetricAxiom(#max(?i), ?A, ?Connector, ?B) :-
    metricAxiom(_, ?i, ?A, ?Connector, ?B) .
```

Here, the predicates `inf:axiom` and `metricTempOp` are the same as before, with `inf:axiom` containing the TBox with redundant axioms. Now, the adjusted TBox can be constructed by checking the predicates in a cascading manner.

```
adjustedAxiom("+-", ?A, ?Connector, ?B) :- plusMinusAxiom(?A, ?Connector, ?B) .
adjustedAxiom("+-", ?A, ?Connector, ?B) :-
    plusAxiom(?A, ?Connector, ?B),
    minusAxiom(?A, ?Connector, ?B),
    ~plusMinusAxiom(?A, ?Connector, ?B).
```

Consider a CI of the form $\diamondsuit A \sqsubseteq B$. The first line checks if there exists an axiom where $\diamondsuit$ is $\boxplus$. If this is the case, the axiom is added to the adjusted TBox. The second line only adds $\boxplus A \sqsubseteq B$ if it does not already exist, but both $\diamondsuit A \sqsubseteq B$ and $\boxminus A \sqsubseteq B$ do. The rest of the operators are checked in a similar manner. The adjusted TBox is then used for the ABox reasoning phase.

3.4 Reading the ABox in

The extraction of the ABox is simpler than the TBox, as the axioms can be added directly as triples. The more interesting challenge, which is discussed later in this section, is the construction of the set of representative time points rep($\mathcal{A}$) from the set tem($\mathcal{A}$). But first the temporal ABox assertions need to be collected.

```
aBoxConcept(?A, ?a, ?i), tem(?i) :-
    triples(?a, rdf:type, owl:NamedIndividual), triples(?A, rdf:type, owl:Class),
    triples(?id, rdf:type, owl:Axiom),
    triples(?id, owl:annotatedSource, ?a), triples(?id, owl:annotatedTarget, ?A),
    triples(?id, ?type, ?i), commentType(?type) .
```

The difference to the construction of the TBox axioms is that the `owl:annotatedSource` is a node of the `rdf:type` `owl:NamedIndividual`. Notice that this rule only collects assertions for concepts and not roles. This distinction is made as the predicate for roles needs one more entry to store the second individual. Otherwise, the construction works the same. Additionally, this rule also collects all the time points which appear in the ABox in the predicate `tem`.

Now, using the accumulated time points, the set rep($\mathcal{A}$) can be constructed. The main idea to find the intervals between neighbouring elements in tem($\mathcal{A}$) is to create a predicate which collects all tuples the left entry is smaller than the right entry. Note that time points may appear multiple times as the left boundary of different intervals. Minimising the right boundary of associated with each left boundary results in the intervals.

The following rule constructs the intervals and their sizes:

```
interval(?k, ?l) :- tem(?k), tem(?l), ?k + 1 <= ?l - 1 .
```

The condition `?k + 1 <= ?l - 1` ensures that the left boundary of the interval is smaller than the right boundary and that the interval is non-empty. Next, the aggregate function `#min` is used to find the smallest interval for each time point:

```
minInterval(?k, #min(?l)) :- interval(?k, ?l) .
```

Now, the set rep($\mathcal{A}$) can be constructed as a predicate.

```
rep(?k, ?k) :- tem(?k) .
rep(?k + 1, ?l - 1) :- minInterval(?k, ?l) .
```

The second line adds the aforementioned intervals of time points which do not appear in $\mathcal{A}$. Additionally, `rep` also includes the elements from `tem` as described in Section 2.3. Observe that the predicate `rep` has an arity of two while rep($\mathcal{A}$) is a simple set. The algorithm would, thus, also work with an unary predicate. However, the next step in the implementation is to make queries on the completed ontology possible. Such a query might include a time point i with $i \notin$ rep($\mathcal{A}$). That is why both boundaries of a represented interval $[k,\ l]$ are stored, as checking if $i \in [k, l]$ holds now consists of two comparisons. The actual representative of $[k, l]$ is just chosen to be the left boundary k.

What remains is to explicitly add the two intervals with $-\infty$ or ∞ as their left, respectively, right boundary.

```
rep(#min(?i) - 1, "-inf") :- tem(?i) .
rep(#max(?i) + 1, "inf") :- tem(?i) .
```

The interval $(-\infty, \min(\text{tem}(\mathcal{A}))-1]$ is the only one for which the right boundary is selected as the representative. Hence, the boundaries are switched in `rep`, such that the representative is in first place. The strings "`inf`" and "`-inf`" are just placeholders for ∞ and $-\infty$. With `rep` finalised, the translation of the ABox rules can be discussed in the next section.

3.5 Translation of the ABox rules

Similarly to the TBox rules in Section 3.3, the translation of the ABox rules A1, A4 and A5 is uncomplicated, which is why only one of them is shown as an example.

As a first example, consider the rule

$$\text{A4}\ \frac{A_1(a,i) \quad A_2(a,i) \quad A_1 \sqcap A_2 \sqsubseteq B}{B(a,i)}\ .$$

It translates into *Nemo's* dialect as follows.

```
inf:aBoxConcept(?B, ?a, ?i) :-
    conj(?idConj, ?A1, ?A2), inf:axiom(_, ?idConj, rdfs:subClassOf, ?B),
    inf:aBoxConcept(?A1, ?a, ?i), inf:aBoxConcept(?A2, ?a, ?i) .
```

The more complicated rules are

$$\text{A2}\ \frac{i \in \diamondsuit_{+}A(a) \quad \diamondsuit_{+}A \sqsubseteq B}{B(a,i)}\ , \qquad \text{A3}\ \frac{i \in \diamondsuit_{+}r(a,b) \quad \diamondsuit_{+}r \sqsubseteq s}{s(a,b,i)}$$

as the involvement of `rep` make a case distinction on the different diamond operators necessary. The first case encompasses all the axioms which involve the $\diamondsuit$ operator.

```
inf:aBoxConcept(?B, ?a, ?i) :-
    inf:axiom("+-", ?A, rdfs:subClassOf, ?B), rep(?i, _),
    inf:aBoxConcept(?A, ?a, _) .
```

This is the simplest case, as it is only necessary to check if the individual `?a` is in `?A` at some point in time. The next two cases comprise the axioms with $\diamondsuit_{+}$ and $\diamondsuit_{-}$, respectively.

```
inf:aBoxConcept(?B, ?a, ?i) :-
    inf:axiom("+", ?A, rdfs:subClassOf, ?B), rep(?i, _),
    inf:aBoxConcept(?A, ?a, ?k), ?i <= ?k .

inf:aBoxConcept(?B, ?a, ?i) :-
    inf:axiom("-", ?A, rdfs:subClassOf, ?B), rep(?i, _),
    inf:aBoxConcept(?A, ?a, ?l), ?i >= ?l .
```

These cases add a check if there exist occurrences of `?a` in `?A` later, respectively earlier, than `?i`. The fourth case is a combination of the previous two, thereby checking for occurrences within arbitrarily-sized intervals. Thus, it encompasses the axioms containing the $\diamondsuit$ operator.

```
inf:aBoxConcept(?B, ?a, ?i) :-
    inf:axiom("con", ?A, rdfs:subClassOf, ?B), rep(?i, _),
    inf:aBoxConcept(?A, ?a, ?k),
    inf:aBoxConcept(?A, ?a, ?l),
    ?i >= ?k, ?i <= ?l .
```

The condition `?i >= ?k` and `?i <= ?l` necessitates that `?i` is between two occurrences of `?a` in `?A`. The last case spans all axioms with metric temporal operators.

```
inf:aBoxConcept(?B, ?a, ?i) :-
    inf:axiom(?op, ?A, rdfs:subClassOf, ?B), rep(?i, _),
    inf:aBoxConcept(?A, ?a, ?k),
    inf:aBoxConcept(?A, ?a, ?l),
    metricTempOp(?op, ?n),
    ?i >= ?k, ?i <= ?l, ?l - ?k < ?n .
```

The addition of `?l - ?k < ?n` imposes that the interval between the occurrences of `?a` in `?A` has at most the size `?n` specified by the operator $\Diamond_n$. The rule A3 works in the same manner and will therefore be omitted here.

This concludes the implementation of the completion algorithm for $\mathcal{TELH}_{\perp}^{\Diamond,\mathbf{lhs}}$ from Borgwardt, Forkel, and Kovtunova [5].

4 A Performance Evaluation

To test the implementation from the previous chapter, a performance evaluation is conducted. This chapter starts by discussing the constructed ontology and the data used for the evaluation. Then, the test setup is presented, followed by the results of the evaluation.

The general approach to the evaluation is to emulate the reasoning process on a database. Therefore, the constructed TBox includes comparatively few axioms, which remain the same during all test runs. The ABox, on the other hand, is generated in different sizes from a large dataset from the real world. This should mimic different workloads in an application.

4.1 Weather Data and the Test Ontology

The main idea behind the constructed ontology is to determine cities in combination with times at which the weather conditions are great for playing beach volleyball. Therefore, the TBox includes some basic rules specifying these conditions, while the ABox is based on weather data from the Open Data Server of the German Meteorological Service (DWD)[1]. The DWD provides, among other services, historical weather data[2] from the last ∼300 years under the Creative Commons licence CC BY 4.0[3]. The data is obtained through the Python package *wetterdienst* [10].

The ontology utilises three types of individuals:

- "City" individuals, which represent a German city where a weather station is located.
- "Temperature" individuals, which are the rounded integer values of the temperature appearing in the ABox.
- Individuals of the form "3379@2024-06-16", where "3379" is the ID of a weather station and "2024-06-16" is a date. These individuals represent a whole day at a specific weather station. They will be referred to as "day/location individuals".

Additionally, the time steps associated with the elements from $\mathbb{Z}$ are 5-minute intervals. Thus, the interpretations $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ represent the state of the atmosphere at two consecutive 5-minute intervals. To populate the ABox, data is taken from the datasets shown in Table 4 [19]. These datasets exist in three versions spanning different time periods. The data is from the "recent" version, which includes recent weather data and the previous 500 days. Every dataset also includes several columns of different data, which is why the DWD's original name of the column is given as well. The DWD also provides forecast data, which would make this ontology

[1] `https://opendata.dwd.de/`

[2] `https://opendata.dwd.de/climate_environment/CDC/`

[3] `https://creativecommons.org/licenses/by/4.0/`

Table 4: The datasets used to populate the test ontology

Dataset	Path on the Open Data Server	Columns used	Description
air_temperature	`/climate_environment/CDC/observations_germany/climate/10_minutes/air_temperature/recent/`	TT_10	Air temperature 2 m above the ground in 10-minute intervals in °C
precipitation	`/climate_environment/CDC/observations_germany/climate/5_minutes/precipitation/recent/`	RS_05	Sum of the precipitation height of the last 5 minutes in mm
wind	`/climate_environment/CDC/observations_germany/climate/10_minutes/wind/recent/`	FF_10	mean wind speed during the previous 10 minutes in m/s
sun	`/climate_environment/CDC/observations_germany/climate/hourly/sun/recent/`	SD_SO	sunshine duration during the last hour in min
cloudiness	`/climate_environment/CDC/observations_germany/climate/hourly/cloudiness/recent/`	V_N	total cloud cover during the last hour in %

more useful in a real-world application. However, the repeatability of the tests would be limited as forecast data changes frequently. Therefore, using past data keeps this evaluation self-contained without having to provide a large dataset of recorded forecast data.

The most important concepts and roles used in the ontology are explained in Table 5. The dataset used to populate the respective predicate is given in Table 4 as well. Additionally to the concepts shown here, the ontology also includes some auxiliary concepts, which are straightforward in the context of their occurrences in axioms. The entire TBox can be seen in Figure 4.

The axioms (ax1) - (ax6) express the interpolation of data which is not provided in 5-minute intervals. For example, the assumption that temperatures of 17 - 25 degrees Celsius are optimal for playing beach volleyball is independent of the current time point. Wind data is only provided in 10-minute intervals. Thus, individuals can at most appear every second $i \in \mathbb{Z}$ in the input ABox. Therefore, (ax2) expresses the assumption that if the wind speed was under 3 m/s at two of these "neighbouring" inputs, it was as well at the moment separating them.

The assumed best weather conditions for playing beach volleyball are as follows. Two dry 5-minute intervals are at most ten minutes apart (ax8), and additionally, the wind speed should be under 3 m/s, making it playable conditions (ax10). The sky should be bright enough but not so bright that it is blinding. Thus, some cloud cover is required (ax9). The temperature should be between 17 and 25 degrees Celsius (ax7, ax7'). The combination of all these conditions is considered the best timefor playing beach volleyball (ax11, ax12). The last two axioms (ax13, ax13') now produce timestamped names of cities where these conditions are met.

The TBox was constructed using Protégé 5.5.0 [18] and is saved in the *Terse RDF Triple Language* (also referred to as *Turtle*).

Table 5: The most important concepts and roles in the test ontology.

	Source Dataset	Description
Concepts:		
OptimalTemperatures	manually added	Includes all temperature individuals which are between 17 and 25 degrees Celsius
NoRain	precipitation	Includes day/location individuals; comprises 5-minute intervals in time where the weather station on the given day did not record any precipitation
Cloudy	cloudiness	Includes day/location individuals; comprises 1-hour intervals in time where the weather station on the given day recorded a cloud cover of more or equal to 30%
Sunshine	sun	Includes day/location individuals; comprises 1-hour intervals in time where the weather station recorded more than one minute of sunshine
WindUnder3	wind	Includes day/location individuals; comprises 10-minute intervals where the wind speed was less than 3 m/s
Roles:		
temperatures	air_temperature	Relates a day/location individual for a 10-minute interval to the individual representing the temperature recorded at the weather station during this interval
dataOfWeatherStation	manually added	Relates a city individual to all day/location individuals which are associated with the weather station located in the city

4.2 The Test Setup

As discussed at the end of Section 3.3 the completion process of the TBox produces redundancies, which can only be removed afterwards. It is, therefore, impossible to quantify the increased reasoning time in this concrete example. However, it is possible to count the number of redundant axioms and measure their effect on the time it takes to complete the ABox. Thus, the completion algorithm is run twice on the same ABox, once with the redundant TBox and once with the adjusted one. The reasoning times are then compared. Additionally, the number of inferred facts and the sizes of the completed ABoxes are logged. On the one hand, this acts as a sanity check, verifying that the reasoning process is working as intended. On the other hand, it enables one to draw comparisons between this implementation and the implementation of the ELK reasoner, as shown in Section 2.5.1.

The evaluation of the ABox reasoning is based on a test suite consisting of a series of fully automated runs of the implemented algorithm on the aforementioned TBox and ABoxes of increasing size. The different sizes are achieved by increasing the time interval within which the weather data is downloaded. The interval starts with both boundaries being the 15th June 2024 00:00 and is then progressively increased in 2-hour steps until the 16th June 2024 22:00, thus creating 24 individual tests. The data is taken from 5 weather stations located in cities all around Germany: Aachen, Hamburg, Dresden, Munich, and Erfurt.

Before the tests of the ABox reasoning start, the TBox is completed once, and it is saved.

(ax1): $\boxplus$OptimalTemperatures $\sqsubseteq$ OptimalTemperatures

(ax2): $\boxplus_4$WindUnder3 $\sqsubseteq$ WindUnder3 (ax3): $\boxplus_{13}$Sunshine $\sqsubseteq$ Sunshine

(ax4): $\boxplus_{13}$Cloudy $\sqsubseteq$ Cloudy (ax5): $\boxplus_3$temperatures $\sqsubseteq$ temperatures

(ax6): $\boxplus$dataOfWeatherStation $\sqsubseteq$ dataOfWeatherStation

(ax7): $\exists$temperatures.OptimalTemperatures $\sqsubseteq$ PleasantTemperatureForBeachvolleyball

(ax7'): $\boxplus_1$PleasantTemperatureForBeachvolleyball $\sqsubseteq$ $\exists$temperatures.OptimalTemperatures

(ax8): $\boxplus_4$NoRain $\sqsubseteq$ DryEnoughForBeachvolleyball

(ax9): Sunshine $\sqcap$ Cloudy $\sqsubseteq$ PleasantSkyForBeachvolleyball

(ax10): DryEnoughForBeachvolleyball $\sqcap$ WindUnder3 $\sqsubseteq$ PossibleToPlayBeachvolleyball

(ax11): PleasantSkyForBeachvolleyball $\sqcap$ PossibleToPlayBeachvolleyball
$\sqsubseteq$ GoodTimeForBeachvolleyball

(ax12): PleasantTemperatureForBeachvolleyball $\sqcap$ GoodTimeForBeachvolleyball
$\sqsubseteq$ BestTimeForBeachvolleyball

(ax13): $\exists$dataOfWeatherStation.BestTimeForBeachvolleyball $\sqsubseteq$ WhereAndWhenToPlay

(ax13'): $\boxplus_1$WhereAndWhenToPlay $\sqsubseteq$ $\exists$dataOfWeatherStation.BestTimeForBeachvolleyball

Figure 4: The test ontology's TBox

Then its redundancies are removed and the adjusted TBox is saved as well.

Each test run consists of four steps. First, the weather data of the according time frame is downloaded and preprocessed. The preparation is done using a Python script, which parses the data and populates the predicates shown in Table 5 according to the conditions written there. After that, the concepts are saved in a CSV file, and the roles are saved in another. This is done as the file with the roles has four columns (role name, temporal operator, two individuals), whereas concepts only need three (concept name, temporal operator, one individual). Second, the *Nemo* program is run on the ABox and the TBox with the redundancies. Third, the same is done with the TBox without redundancies. Finally, the reasoning times, the number of overall inferred facts, and the sizes of the completed ABoxes are logged. These 25 tests (TBox reasoning + 24 ABox reasoning tests) are run three times and the resulting reasoning times for each input size are averaged. This should eliminate outliers and provide a more reliable result.

A more granular view of the reasoning process can be provided using the Python bindings of *Nemo*, as they allow access to the reasoning times on a per-rule basis. Thus, it can be analysed what portion of the execution time each rule takes. Although the Python bindings could be used to invoke the reasoning process, earlier testing has shown that they are a lot slower than the *Nemo* command-line client. Therefore, during the performance evaluation, the Python script invokes the *Nemo* command-line client, and the reasoning times of each rule are only analysed for the single ABox with 927 facts. This ABox covers the weather data between 00:00 and 10:00 on the 15th June 2024 in Dresden and is chosen as an arbitrary example.

The tests were run on a desktop PC with an AMD Ryzen Octa-Core processor and 32 GB of RAM. The operating system used was Windows 11 Pro. *Nemo* version 0.5.2-dev (commit f55fe02d) was used for both the command-line and the Python bindings. They were compiled using *rustc* version 1.78.0 and *cargo* version 1.78.0. The Python version used was 3.12.4.

4.3 Discussion of Results

Before discussing the reasoning times measured during the run of the test suite, the results of the reasoning process should be mentioned shortly. The ontology actually contains 98 times and locations at which the conditions were optimal to play beach volleyball after its completion. The conditions were right in Munich, Hamburg and Dresden, and all the suggested times are between 10:00 and 19:00, which is realistic.

The completion of the TBox took, on average, 4 ms. This short time is rather unsurprising, as the TBox consists of only 14 axioms. The completed TBox contains 43 axioms, including six redundant ones. Four of them are versions of (ax1) - (ax4) with the $\Diamond_1$ operator created by T1. The other two axioms are versions of (ax5) and (ax6) also with the $\Diamond_1$ operator created by T3. These redundancies are expected.

Figure 5 shows the number of facts in the completed ABox in addition to the number of overall inferred facts during the algorithm's run in relation to the number of facts in the input ABox. The first insight is that the completion algorithm produces the same-sized output independent of the redundancies in the TBox, which indicates that the algorithm is working as intended.

The chart shows a clear linear relationship between the size of the input ABox and the size of the completed ABox. The trend line $f(x) = 8.794x - 592.8$ is obtained using the least squares method. The fit, as displayed in the chart, is good.

The relationship between the number of overall inferred facts and the size of the input ABox follows a polynomial of a higher degree. Using the least squares method, polynomials of different degrees can be fitted to the data. But already the resulting cubic function $f(x) = -8.624 \cdot 10^{-8} \cdot x^3 + 0.007x^2 + 10.53x - 827.5$ shows a minuscule first coefficient, indicating that the relationship is quadratic. Fitting a second-degree polynomial to the data using the least squares method produces the following equation: $f(x) = 0.006596x^2 + 11.64x - 1239$. The coefficient of the last term being -1239 is questionable, as it does not make sense to start

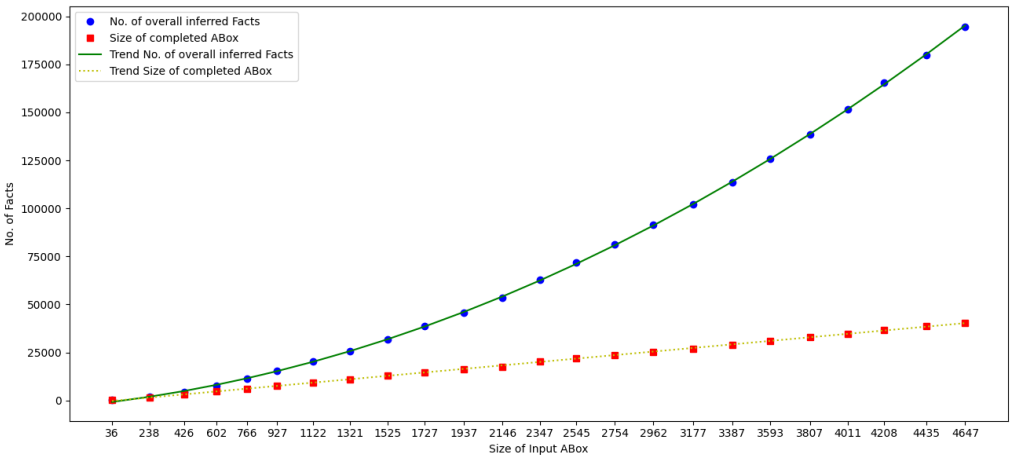


Figure 5: The number of inferred ABox facts in relation to the size of the input ABox.

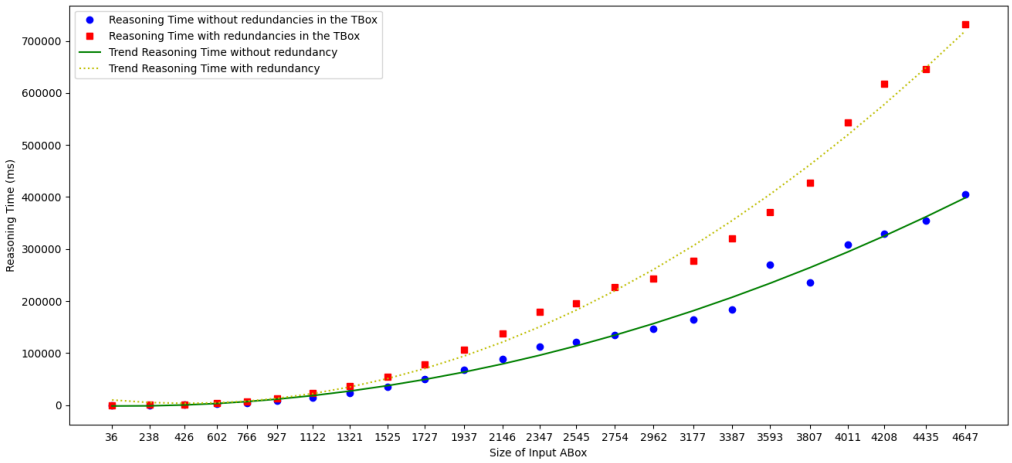


Figure 6: The reasoning times in relation to the size of the input ABox.

with a negative number of facts. But the trend line, as displayed in Figure 5, shows a good fit.

The resulting times from the test suite are shown in Figure 6. First, notice that the data is less smooth than in Figure 5. Although the times were averaged from three runs, there still exist outliers. This is in part due to an increasing spread of the runtimes the larger the input ABox is. For example, two runs with the largest input ABox size of 4647 facts took 681 and 690 seconds, respectively, while a third run took 822 seconds. The reason for this is not clear, as these runs happened under the same conditions without any other programs running in the background.

Even with outliers, the general trend is clear. The reasoning times with redundant axioms in the TBox are quite a bit higher than the ones without. This is surprising, as the redundancies are only very few and only consist of trivial axioms. Using the least squares method the polynomials seen in Table 6 are obtained.

Table 6: The polynomials fitted to the reasoning times in relation to the size of the input ABox.

	Quadratic Polynomial	Cubic Polynomial
Reasoning Times with redundancies in the TBox	$0.04x^2 - 35.52x + 1.113 \cdot 10^4$	$4.419 \cdot 10^{-6} \cdot x^3 + 0.009x^2 + 21.73x - 9965$
Reasoning Times without redundancies in the TBox	$0.0194x^2 - 4.024x - 1585$	$3.95 \cdot 10^{-7} \cdot x^3 + 0.0167x^2 + 1.093x - 3470$

Both datasets show the behaviour that the coefficient for the third-degree term is not significant. Thus, the quadratic polynomials are shown in Figure 6. Again, the terms of zeroth degree do not make sense as there should not exist a negative reasoning time, and the reasoning time should also not start at 11 seconds when the ABox is empty. This cannot be constant overhead, as small ABoxes take less than a second to be completed. Otherwise, their trends look like they fit the data well. Both polynomials being quadratic also follows the trend of the overall inferred facts, which makes sense. The quadratic trend can be explained by the application of rules A2 and A3 (see Section 3.3), where the algorithm needs to check if there exists a certain interval within which a certain individual (or pair of individuals) appears in a concept (or a role). Therefore, the algorithm might need to check all possible combinations of elements from the predicate `rep`.

Quantitatively, the reasoning with redundant axioms in the TBox took, on average, 1.62 times longer than the reasoning without. Similarly to the spread of the data, the ratio of the reasoning times was not constant but instead fluctuated within the range of 1.47 to 1.87 with a tendency to increase with a bigger input. This shows that it is crucial to remove redundancies from the TBox before reasoning with the ABox.

Overall, the performance of this implementation is promising but not optimal. Completing the largest input, ABox takes about 400s while deriving about 194 000 facts. The implementation of the rules from the ELK reasoner in *Nemo*, on the other hand, takes about 4 seconds on the same hardware to derive 2.4 million facts from an input of about 246 000 axioms.

The per-rule analysis of reasoning times revealed that about a third of the time was spent applying each of the following two rules.

```
1       inf:aBoxConcept(?B, ?a, ?i) :-
2           inf:axiom(?op, ?A, rdfs:subClassOf, ?B), rep(?i, _),
3           inf:aBoxConcept(?A, ?a, ?k),
4           inf:aBoxConcept(?A, ?a, ?l),
5           metricTempOp(?op, ?n),
6           ?i >= ?k, ?i <= ?l, ?l - ?k < ?n .
7       inf:aBoxRole(?S, ?a, ?b, ?i) :-
8           inf:axiom("con", ?R, rdfs:subPropertyOf, ?S), rep(?i, _),
9           inf:aBoxRole(?R, ?a, ?b, ?k),
10          inf:aBoxRole(?R, ?a, ?b, ?l),
11          ?i >= ?k, ?i <= ?l .
```

This indicates that the conditions $i \in \Diamond\!\!\!\!-\, A(a)$ and $i \in r(a, b)$ from rules A2 and A3 (see Section 2.3) are bottlenecks in the reasoning process. For convex diamonds, the implemented algorithm needs to check if an interval $[j, k]$ with $i \in [j, k]$, and $j, k \in \text{rep}(\mathcal{A})$, and $A(a, j), A(a, k) \in \mathcal{A}$ exists (similarly for $r(a, b, i)$). Therefore, it may check all combinations of two elements within $\text{rep}(\mathcal{A})$, which is computationally intensive.

The reasons that these cases of the rules A2 and A3 take an especially long time to execute are probably the following. The first rule is applied a lot as the TBox includes a lot of axioms where a metric convex diamond is applied to a concept. It can be speculated that the second rule takes so much computation time because the entries in dataOfWeatherStation are only provided in the first and last 5-minute intervals of the day. Thus, the algorithm has to interpolate the missing information for the rest of the day because of (ax6) and, therefore, adds 286 new entries per day and weather station. These new entries increase the number of combinations of `inf:aBoxRole(?R, ?a, ?b, ?k)` and `inf:aBoxRole(?R, ?a, ?b, ?l)`. This introduces more possible checks, which slows the reasoning process down. Thus, the most productive optimisation of the algorithm would probably be to improve these checks.

An idea would be to store the *intervals* in which an individual occurs in a concept instead of each time point. This would reduce the number of entries in the `aBoxConcept` and `aBoxRole` predicates, as then at most $|\texttt{rep}|/2$ entries per combination of concept and individual or role and pair of individuals are needed. This would be the case if the individual alternates between occurring and not occurring in the concept. However, these intervals might be subject to change during the reasoning process as additional occurrences are derived. Therefore, intervals and thus entries in `aBoxConcept` and `aBoxRole`, would need to be modified or fused, which is currently not possible in *Nemo*. Hence, the ABox would include a large number of redundant facts, which would negate the advantage of using intervals to store the occurrences.

5 Conclusion

In this thesis, the reasoning algorithm for $\mathcal{TELH}_{\bot}^{\Diamond\text{-lhs}}$ from Borgwardt, Forkel, and Kovtunova [5] was implemented in the Datalog-based rule engine *Nemo*. It allows the completion of temporalised OWL ontologies. For this purpose, the extension of OWL with a new annotation property to encode temporal information was proposed. Additionally, the necessary preprocessing steps were discussed. The construction of the order on the diamond operators and the set of representative time points was essential. Having constructed these crucial predicates, the translation of the rules was straightforward.

The implementation was then evaluated on the completion of a synthetic ontology based on weather data. The size of the input ABox was varied and the reasoning times were measured. Additionally, the impact of redundancies in the TBox, which arise during the reasoning process, was analysed. The evaluation revealed that removing redundancies from the TBox is crucial, as they significantly raise the reasoning time by a factor of 1.62 on average. Furthermore, an analysis of the portions of the overall reasoning time each individual rule took to execute demonstrated that the conditions $i \in \Diamond A(a)$ and $i \in \Diamond r(a, b, i)$ in rules A2 and A3 are bottlenecks for the implementation's performance. Overall, the evaluation showed promising results, but the implementation is currently not viable for working on extensive datasets as it is significantly slower than other established reasoners for similar, although atemporal DLs like ELK [13].

Therefore, in future work, this implementation could be revisited and revised once *Nemo* supports the modification and deletion of existing entries in predicates. Optimisations like the different encoding of time points suggested at the end of the previous chapter could be investigated. Another fruitful approach might be to apply the same ideas used to optimise the rules in the ELK reasoner [13] to improve the TBox rules, as this should also improve the performance if a larger TBox is input.

Furthermore, this implementation should be extended with a preprocessing step that normalises the input ontology so that the users do not have to provide a normalised ontology themselves. This task is not trivial because OWL 2 only supports annotation properties on the granularity of axioms, not on the granularity of the individual concepts or roles within the axiom. Therefore, first a new way to encode the temporal information in OWL 2 ontologies to accommodate nested diamond operators would need to be proposed. However, this would make the algorithm more accessible and, therefore, more viable in real-world applications.

Finally, a temporal extension of Datalog called *DatalogMTL* exists, which features metric temporal operators. However, they are not convex [20]. Therefore, the relationship between $\mathcal{TELH}_{\bot}^{\Diamond\text{-lhs}}$ and DatalogMTL could be investigated.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN: 0-201-53771-0. URL: http://webdam.inria.fr/Alice/.

[2] A. Artale, R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. "Temporalising Tractable Description Logics". In: *14th International Symposium on Temporal Representation and Reasoning (TIME'07)*. 14th International Symposium on Temporal Representation and Reasoning (TIME'07). Alicante, Spain: IEEE, 2007, pp. 11–22. DOI: 10.1109/TIME.2007.62.

[3] Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. "Ontology-Mediated Query Answering over Temporal Data: A Survey (Invited Talk)". In: *LIPIcs, Volume 90, TIME 2017* 90 (2017). In collab. with Sven Schewe, Thomas Schneider, and Jef Wijsen. Artwork Size: 37 pages, 895326 bytes ISBN: 9783959770521 Medium: application/pdf Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1:1–1:37. DOI: 10.4230/LIPICS.TIME.2017.1.

[4] F. Baader, S. Brandt, and C. Lutz. "Pushing the $\mathcal{EL}$ Envelope". In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*. Edinburgh, UK: Morgan-Kaufmann Publishers, 2005.

[5] Stefan Borgwardt, Walter Forkel, and Alisa Kovtunova. "Finding New Diamonds: Temporal Minimal-World Query Answering over Sparse ABoxes". In: *Rules and Reasoning*. Ed. by Paul Fodor, Marco Montali, Diego Calvanese, and Dumitru Roman. Cham: Springer International Publishing, 2019, pp. 3–18. DOI: 10.1007/978-3-030-31095-0_1.

[6] Diego Calvanese, Jeremy Carroll, Guiseppe De Giacomo, Jim Hendler, Ivan Herman, Parsia Bijan, Peter F. Patel-Schneider, Alan Ruttenberg, Uli Sattler, and Michael Schneider. "OWL 2 Web Ontology Language Profiles (Second Edition)". In: (2012). Ed. by Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. URL: https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[7] S. Ceri, G. Gottlob, and L. Tanca. "What you always wanted to know about Datalog (and never dared to ask)". In: *IEEE Transactions on Knowledge and Data Engineering* 1.1 (Mar. 1989), pp. 146–166. DOI: 10.1109/69.43410.

[8] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. "Complexity and expressive power of logic programming". In: *ACM Comput. Surv.* 33.3 (Sept. 2001). Place: New York, NY, USA Publisher: Association for Computing Machinery, pp. 374–425. DOI: 10.1145/502807.502810.

[9] Víctor Gutiérrez-Basulto, Jean Christoph Jung, and Roman Kontchakov. "Temporalized EL ontologies for accessing temporal data: complexity of atomic queries". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. Place: New York, New York, USA. AAAI Press, 2016, pp. 1102–1108.

[10] Benjamin Gutzmann and Andreas Motl. *wetterdienst*. Version v0.91.0. July 2024. DOI: 10.5281/zenodo.12739518.

[11] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. 1st Edition. Chapman and Hall/CRC, 2009. 456 pp. URL: https://learning.oreilly.com/library/view/foundations-of-semantic/9781420090512/.

[12] Alex Ivliev, Stefan Ellmauthaler, Lukas Gerlach, Maximilian Marx, Matthias Meißner, Simon Meusel, and Markus Krötzsch. "Nemo: First Glimpse of a New Rule Engine". In: *Proceedings 39th International Conference on Logic Programming (ICLP 2023)*. Ed. by Enrico Pontelli, Stefania Costantini, Carmine Dodaro, Sarah Gaggl, Roberta Calegari, Artur D'Avila Garcez, Francesco Fabiano, Alessandra Mileo, Alessandra Russo, and Francesca Toni. Vol. 385. EPTCS. Sept. 2023, pp. 333–335. DOI: 10.4204/EPTCS.385.35.

[13] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. "The Incredible ELK". In: *Journal of Automated Reasoning* 53.1 (June 1, 2014), pp. 1–61. DOI: 10.1007/s10817-013-9296-3.

[14] Bas Ketsman and Paraschos Koutris. "Modern Datalog Engines". In: *Foundations and Trends® in Databases* 12.1 (2022), pp. 1–68. DOI: 10.1561/1900000073.

[15] Markus Krötzsch. "Database Theory - Lecture 18: The Chase". 2022. URL: https://iccl.inf.tu-dresden.de/w/images/4/43/DBT2022-Lecture-18-overlay.pdf.

[16] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. "A Description Logic Primer". In: *CoRR* abs/1201.4089 (2012). arXiv: 1201.4089. URL: http://arxiv.org/abs/1201.4089.

[17] Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. "Temporal Description Logics: A Survey". In: *2008 15th International Symposium on Temporal Representation and Reasoning*. 2008 15th International Symposium on Temporal Representation and Reasoning (TIME '08). Montreal, QC: IEEE, June 2008, pp. 3–14. DOI: 10.1109/TIME.2008.14.

[18] Mark A. Musen. "The protégé project: a look back and a look forward". In: *AI Matters* 1.4 (June 16, 2015), pp. 4–12. DOI: 10.1145/2757001.2757003.

[19] *Open Data Server of the Deutscher Wetterdienst*. Germany. URL: https://opendata.dwd.de/climate_environment/CDC/.

[20] Przemysław A. Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. "DatalogMTL: Computational Complexity and Expressive Power". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Twenty-Eighth International Joint Conference on Artificial Intelligence {IJCAI-19}. Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 1886–1892. DOI: 10.24963/ijcai.2019/261.