# Revising Description Logic Terminologies to Handle Exceptions: a First Step

Roberto Micalizio, Gian Luca Pozzato

Dipartimento di Informatica - Università degli Studi di Torino
roberto.micalizio@unito.it,gianluca.pozzato@unito.it

**Abstract.** We propose a methodology to revise a Description Logic knowledge base when detecting exceptions. Our approach relies on the methodology for debugging a Description Logic terminology, addressing the problem of diagnosing incoherent ontologies by identifying a minimal subset of axioms responsible for an inconsistency. In the approach we propose, once the source of the inconsistency has been localized, the identified axioms are revised in order to obtain a consistent knowledge base including the detected exception. To this aim, we make use of a nonmonotonic extension of the Description Logic $\mathcal{ALC}$ based on the combination of a typicality operator and the well established nonmonotonic mechanism of rational closure, which allows to deal with prototypical properties and defeasible inheritance.

## 1 Introduction

*Exceptions* do exist. Intuitively, we can define an exception as an individual (or a group of individuals) that has, or has not, a property in contrast with other individuals of the same class. For instance, a *penguin* can be considered as an exceptional *bird* since, although equipped with wings, is unable to fly. We can find numerous examples of exceptions in nature, but also in natural languages (e.g., irregular verbs), medicine (e.g. *situs inversus* is an anomaly in which the major visceral organs are reversed or mirrored from their normal positions, so that the heart is positioned in the right-hand side of the chest), and many other real-world scenarios.

Dealing with exceptions can be tricky for an ontology engineer. All the exceptions should be known in advance by the ontology engineer; i.e., when the ontology is being designed. Unfortunately, in many real-world scenarios, exceptions are discovered just while the system is running, and the ontology is actually used. Whenever exceptions are discovered at runtime, the resulting ontology becomes incoherent (i.e., at least one concept is mapped to an empty set of individuals). Some recent works [16, 17, 15, 8, 9, 13] have addressed the problem of diagnosing incoherent ontologies by identifying a minimal subset of axioms responsible for the inconsistency. The idea of these works is that once the source of the inconsistency has been localized, the ontology engineer can intervene and revise the identified axioms in order to restore the consistency. These approaches presuppose that the ontology has become incoherent due to the introduction of *errors*; as for instance when two ontologies are merged together. It is worth noting that, albeit an exception has the same effect of an error (i.e., it causes an ontology to become incoherent), an exception is not an error. An exception is rather a piece of knowledge that partially contradicts what is so far known about a portion of the ontology at hand. Thus, on the one hand, ignoring exceptions would be deleterious as the resulting ontology would not reflect the applicative domain correctly. On the other hand, accommodating exceptions requires the exploitation of some form of defeasible reasoning that allows us to revise some of concepts in the ontology.

In this paper we propose a methodology to revise a Description Logic (for short: DL) knowledge base when detecting exceptions. Our approach relies on the above mentioned methodology of [16, 17, 15] for detecting exceptions by identifying a minimal subset of axioms responsible for an inconsistency. Once the source of the inconsistency has been localized, the identified axioms are revised in order to obtain a consistent knowledge base including the detected exception. To this aim, we use a nonmonotonic extension of the DL $\mathcal{ALC}$ recently presented by Giordano and colleagues in [7]. This extension is based on the introduction of a typicality operator $\mathbf{T}$ in order to express typical inclusions. The intuitive idea is to allow concepts of the form $\mathbf{T}(C)$, whose intuitive meaning is that $\mathbf{T}(C)$ selects the *typical* instances of a concept $C$. It is therefore possible to distinguish between properties holding for all instances of a concept $C$ ($C \sqsubseteq D$), and those only holding for the typical instances of $C$ ($\mathbf{T}(C) \sqsubseteq D$). For instance, a knowledge base can consistently express that birds normally fly ($\mathbf{T}(\mathit{Bird}) \sqsubseteq \mathit{Fly}$), but penguins are exceptional birds that do not fly ($\mathit{Penguin} \sqsubseteq \mathit{Bird}$ and $\mathit{Penguin} \sqsubseteq \neg\mathit{Fly}$).

The $\mathbf{T}$ operator is intended to enjoy the well-established properties of *rational logic*, introduced by Lehmann and Magidor in [12] for propositional logic. In order to reason about prototypical properties and defeasible inheritance, the semantics of this nonmonotonic DL, called $\mathcal{ALC}^{\mathbf{R}}_{min}\mathbf{T}$, is based on rational models and exploits a minimal models mechanism based on the minimization of the rank of domain elements. This semantics corresponds to a natural extension to DLs of Lehmann and Magidor's notion of *rational closure* [12].

The paper is organized as follows: in Section 2 we recall extensions of DLs for prototypical reasoning, focusing on the approach based on a typicality operator $\mathbf{T}$ and rational closure [7]; in Section 3 we recall and extend the notions introduced in [17, 13] for diagnosing incoherent ontologies; in Section 4 we present our methodology for revising a DL terminology to handle exceptions by means of $\mathbf{T}$; we conclude with some pointers to future issues in Section 5.

## 2 Description Logics and Exceptions

The family of Description Logics [1] is one of the most important formalisms of knowledge representation. DLs are reminiscent of the early semantic networks and of frame-based systems. They offer two key advantages: (i) a well-defined semantics based on first-order logic and (ii) a good trade-off between expressivity and computational complexity. DLs have been successfully implemented by a range of systems, and are at the base of languages for the Semantic Web such as OWL. In a DL framework, a knowledge base (KB) comprises two components: (i) a **TBox**, containing the definition of concepts (and possibly roles) and a specification of inclusion relations among them; (ii) an **ABox**, containing instances of concepts and roles, in other words, properties and relations of individuals. Since the primary objective of the **TBox** is to build a taxonomy of concepts, the need of representing prototypical properties and of reasoning about defeasible inheritance of such properties easily arises.

In the recent years, a large amount of work has been done in order to extend the basic formalism of DLs with nonmonotonic reasoning features. The traditional approach is to handle defeasible inheritance by integrating some kind of nonmonotonic reasoning mechanisms [5, 2, 10, 3, 4]. A simple but powerful nonmonotonic extension of DLs is proposed in [6]. In this approach, "typical" or "normal" properties can be directly specified by means of a "typicality" operator $\mathbf{T}$ enriching the underlying DL; the typicality operator $\mathbf{T}$ is essentially characterized by the core properties of nonmonotonic reasoning, axiomatized by *preferential logic* $\mathbf{P}$ in [11].

In this work we refer to the most recent approach proposed in [7], where the authors extend $\mathcal{ALC}$ with $\mathbf{T}$ by considering *rational closure* as defined by Lehman and Magidor [12] for propositional logic. Here the $\mathbf{T}$ operator is intended to enjoy the well-established properties of rational logic $\mathbf{R}$ . Even if $\mathbf{T}$ is a nonmonotonic operator (so that for instance $\mathbf{T}(\mathit{Bird}) \sqsubseteq \mathit{Fly}$ does not entail that $\mathbf{T}(\mathit{Bird} \sqcap \mathit{Penguin}) \sqsubseteq \mathit{Fly}$), the logic itself is monotonic. Indeed, in this logic it is not possible to monotonically infer from $\mathbf{T}(\mathit{Bird}) \sqsubseteq \mathit{Fly}$, in the absence of information to the contrary, that also $\mathbf{T}(\mathit{Bird} \sqcap \mathit{Black}) \sqsubseteq \mathit{Fly}$. Nor it can be nonmonotonically inferred from $\mathit{Bird}(\mathit{tweety})$, in the absence of information to the contrary, that $\mathbf{T}(\mathit{Bird})(\mathit{tweety})$. Nonmonotonicity is achieved by adapting to $\mathcal{ALC}$ with $\mathbf{T}$ the propositional construction of rational closure. This nonmonotonic extension allows to infer typical subsumptions from the TBox. Intuitively, and similarly to the propositional case, the rational closure construction amounts to assigning a *rank* (a level of exceptionality) to every concept; this rank is used to evaluate typical inclusions of the form $\mathbf{T}(C) \sqsubseteq D$: the inclusion is supported by the rational closure whenever the rank of $C$ is strictly smaller than the rank of $C \sqcap \neg D$. From a semantic point of view, nonmonotonicity is achieved by defining, on the top of $\mathcal{ALC}$ with typicality, a minimal model semantics where the notion of minimality is based on the minimization of the ranks of the domain elements. The problem of extending rational closure to **ABox** reasoning is also taken into account: in order to ascribe typical properties to individuals, the typicality of an individual is maximized. This is done by minimizing its rank (that is, its level of exceptionality). Let us recall the resulting extension $\mathcal{ALC}^{\mathbf{R}}_{min}\mathbf{T}$ in detail.

**Definition 1.** *We consider an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individual constants $\mathcal{O}$. Given $A \in \mathcal{C}$ and $R \in \mathcal{R}$, we define:*
$C_R := A \mid \top \mid \bot \mid \neg C_R \mid C_R \sqcap C_R \mid C_R \sqcup C_R \mid \forall R.C_R \mid \exists R.C_R$
$C_L := C_R \mid \mathbf{T}(C_R)$
*A knowledge base is a pair (TBox, ABox). TBox contains a finite set of concept inclusions $C_L \sqsubseteq C_R$. ABox contains assertions of the form $C_L(a)$ and $R(a, b)$, where $a, b \in \mathcal{O}$.*

We define the semantics of the *monotonic* $\mathcal{ALC} + \mathbf{T_R}$, formulated in terms of rational models: ordinary models of $\mathcal{ALC}$ are equipped with a *preference relation*

$<$ on the domain, whose intuitive meaning is to compare the "typicality" of domain elements, that is to say $x < y$ means that $x$ is more typical than $y$. Typical members of a concept $C$, that is members of $\mathbf{T}(C)$, are the members $x$ of $C$ that are minimal with respect to this preference relation (s.t. there is no other member of $C$ more typical than $x$).

**Definition 2 ([7]).** *A model $\mathcal{M}$ of $\mathcal{ALC}+\mathbf{T_R}$ is any structure $\langle \Delta, <, I\rangle$ where: $\Delta$ is the domain; $<$ is an irreflexive, transitive and modular (if $x < y$ then either $x < z$ or $z < y$) relation over $\Delta$; $I$ is the extension function that maps each concept $C$ to $C^I \subseteq \Delta$, and each role $R$ to $R^I \subseteq \Delta \times \Delta$ as follows: $\top^I = \Delta$, $\bot^I = \emptyset$, $(\neg C)^I = \Delta \backslash C^I$, $(C \sqcap D)^I = C^I \cap D^I$, $(C \sqcup D)^I = C^I \cup D^I$, $(\forall R.C)^I = \{x \in \Delta \mid \forall y.(x, y) \in R^I \to y \in C^I\}$, $(\exists R.C)^I = \{x \in \Delta \mid \exists y.(x, y) \in R^I$ and $y \in C^I\}$, $(\mathbf{T}(C))^I = Min_<(C^I)$, where $Min_<(S) = \{u : u \in S$ and $\nexists z \in S$ such that $z < u\}$. Furthermore, $<$ satisfies the $Well-Foundedness\ Condition$, i.e., for all $S \subseteq \Delta$, for all $x \in S$, either $x \in Min_<(S)$ or $\exists y \in Min_<(S)$ such that $y < x$.*

Given an $\mathcal{ALC}+\mathbf{T_R}$ model $\mathcal{M}= \langle \Delta, <, I\rangle$, we say that (i) $\mathcal{M}$ satisfies an inclusion $C \sqsubseteq D$ if it holds $C^I \subseteq D^I$, (ii) $\mathcal{M}$ satisfies an assertion $C(a)$ if $a^I \in C^I$ and (iii) $\mathcal{M}$ satisfies an assertion $R(a, b)$ if $(a^I, b^I) \in R^I$. Given $K$=(TBox,ABox), we say that $\mathcal{M}$ satisfies TBox if $\mathcal{M}$ satisfies all inclusions in TBox, $\mathcal{M}$ satisfies ABox if $\mathcal{M}$ satisfies all assertions in ABox and $\mathcal{M}$ satisfies $K$ if it satisfies both its TBox and its ABox.

Given a knowledge base $K$, an inclusion $C_L \sqsubseteq C_R$ and an assertion $C_L(a)$, with $a \in \mathcal{O}$, we say that the inclusion $C_L \sqsubseteq C_R$ is derivable from $K$, written $K \models_{\mathcal{ALC}^{\mathbf{R}}\mathbf{T}} C_L \sqsubseteq C_R$, if ${C_L}^I \subseteq {C_R}^I$ holds in all models $\mathcal{M} = \langle \Delta, <, I\rangle$ satisfying $K$. Moreover, we say the assertion $C_L(a)$ is derivable from $K$, written $K \models_{\mathcal{ALC}^{\mathbf{R}}\mathbf{T}} C_L(a)$, if $a^I \in {C_L}^I$ holds in all models $\mathcal{M} = \langle \Delta, <, I\rangle$ satisfying $K$.
As already mentioned, although the typicality operator $\mathbf{T}$ itself is nonmonotonic (i.e. $\mathbf{T}(C) \sqsubseteq D$ does not imply $\mathbf{T}(C \sqcap E) \sqsubseteq D$), the logic $\mathcal{ALC}+\mathbf{T_R}$ is monotonic: what is inferred from $K$ can still be inferred from any $K'$ with $K \subseteq K'$. This is a clear limitation in DLs. As a consequence of the monotonicity of $\mathcal{ALC}+\mathbf{T_R}$, one cannot deal with irrelevance, for instance. So, from the knowledge base of birds and penguins, one cannot derive that $K \models_{\mathcal{ALC}^{\mathbf{R}}\mathbf{T}} \mathbf{T}(\mathit{Penguin} \sqcap \mathit{Black}) \sqsubseteq \neg \mathit{Fly}$, even if the property of being black is irrelevant with respect to flying. In the same way, if we added to $K$ the information that Tweety is a bird ($\mathit{Bird}(\mathit{tweety})$), in $\mathcal{ALC}+\mathbf{T_R}$ one cannot tentatively derive, in the absence of information to the contrary, that $\mathbf{T}(\mathit{Bird})(\mathit{tweety})$ and $\mathit{Fly}(\mathit{tweety})$.

In order to tackle this problem, in [7] the definition of rational closure introduced by Lehmann and Magidor [12] for the propositional case has been extended to the DL $\mathcal{ALC}+\mathbf{T_R}$. Due to space limitations, we omit all definitions of the rational closure of a DL knowledge base, reminding to [7].

From a semantic point of view, in [7] it is shown that minimal rational models that minimize the rank of domain elements can be used to give a semantical reconstruction of this extension of rational closure. The rank $k_{\mathcal{M}}$ of a domain element $x$ is the length of the longest chain $x_0 < \cdots < x$ from $x$ to a minimal $x_0$ (i.e. such that there is no $x'$ such that $x' < x_0$). The idea is as follows: given two models of $K$, one in which a given domain element $x$ has rank $x_1$ and another in which it has rank $x_2$, with $x_1 > x_2$, then the latter is preferred, as in this model the element $x$ is "more normal" than in the former.

Given a knowledge base $K$=(**TBox**,**ABox**), in [7] it is shown that an inclusion $C \sqsubseteq D$ (respectively, an assertion $C(a)$) belongs to the rational closure of $K$ if and only if $C \sqsubseteq D$ (resp., $C(a)$) holds in all minimal models of $K$ of a "special" kind, named *canonical models*. The rational closure construction for $\mathcal{ALC}$ is inexpensive, since it retains the same complexity of the underlying logic, and thus a good candidate to define effective nonmonotonic extensions of DLs. More precisely, the problem of deciding whether a typical inclusion belongs to the rational closure of the **TBox** is in ExpTime as well as the problem of deciding whether an assertion $C(a)$ belongs to the rational closure over the **ABox**.

## 3 Explaining Incoherent Terminologies

In this paper, we propose a methodology to deal with exceptions that builds up on the methodology for debugging a DL terminology proposed by Schlobach et al. since their seminal work [17].

Let us first introduce the notion of incoherent terminology. Given a TBox $\mathcal{T}$, we say that it is coherent if there is no unsatisfiable concept, in other words there is at least a model of $\mathcal{T}$ in which the extensions of all concepts are not empty.

To explain incoherences in terminologies, Schlobach et al. propose a methodology based on two steps: first, *axiom pinpointing* excludes axioms which are irrelevant to the incoherence; second, *concept pinpointing* provides a simplified definition highlighting the exact position of a contradiction within the axioms previously selected. In this paper we are interested in the axiom pinpointing step, which identifies debugging-relevant axioms. Intuitively, an axiom is relevant for debugging if, when removed, a **TBox** becomes coherent, or at least one previously unsatisfiable concept turns satisfiable. The notion of subset of relevant axioms is captured by the following definition.

**Definition 3 (MUPS, Definition 3.1 [17]).** *Let $C$ be a concept which is unsatisfiable in a* **TBox** *$\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a minimal unsatisfiability-preserving sub-TBox (MUPS) of $\mathcal{T}$ if $C$ is unsatisfiable in $\mathcal{T}'$, and $C$ is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$.*

In the following, *mups*$(\mathcal{T}, C)$ is used to denote the set of MUPS for a given terminology $\mathcal{T}$ and a concept $C$. Intuitively, each set of axioms in *mups*$(\mathcal{T}, C)$ represents a conflict set; i.e., a set of axioms that cannot all be satisfied. From this point of view, it is therefore possible to infer a diagnosis for the concept $C$ by applying the Hitting-Set tree algorithm proposed by Reiter [14]. However, the set *mups*$(\mathcal{T}, C)$ is sufficient for our purpose of dealing with exceptions.

A drawback of the debugging approach in [16, 17] is that it is restricted to *unfoldable* TBoxes, only containing unique, acyclic definitions. An axiom is called a definition of $A$ if it is of the form $A \sqsubseteq C$, where $A \in \mathcal{C}$ is an atomic concept. An axiom $A \sqsubseteq C$ is unique if the KB contains no other definition of $A$. An axiom is acyclic if $C$ does not refer either directly or indirectly (via other axioms) to $A$ [1]. This restriction is too strong for our objective of representing and reasoning about defeasible inheritance in a natural way. As an example, the TBox expressing that students are not tax payers, but working students do pay taxes, can be naturally expressed by the following, not unfoldable, TBox=$\{Student \sqsubseteq \neg TaxPayer, Student \sqcap Worker \sqsubseteq TaxPayer\}$. In order to overwhelm this gap, in [13, 8] axiom pinpointing is extended to general TBoxes, more suitable to our purposes. As a further difference, Schlobach and colleagues limit their attention to the basic $\mathcal{ALC}$, whereas Parsia and colleagues in [8] are able to deal also with the more expressive $\mathcal{SHOIN}$, corresponding to OWL-DL. A set of algorithms for computing axiom pinpointing, in particular to compute the set of MUPS for a given terminology $\mathcal{T}$ and a concept $A$, is also provided.

In [16], Schlobach also proposes a methodology for explaining concept subsumptions. The idea is to reduce the structural complexity of the original concepts in order to highlight the logical interplay between them. To this aim, Schlobach proposes to exploit the structural similarity of concepts, that can be used to simplify terminological concepts, and hence the subsumption relations. The structural similarity is based on the notion of *qualified subconcepts*; namely, variants of those concepts a knowledge engineer explicitly uses in the modeling process, and where the context (i.e., sequence of quantifiers and number of negations) of this use is kept intact. Schlobach specifies the notion of qualified subconcepts in two ways: Generalized Qualified Subconcepts ($gqs$), and Specialized Qualified Subconcepts ($sqs$) which are defined by induction as follows:

**Definition 4 (Generalized and Specialized Qualified Subconcepts, [16]).** *Given concepts A, C and D, we define:*

$$\begin{array}{ll}
gqs(A) = sqs(A) = \{A\} & \textit{if } A \textit{ is atomic}\\
gqs(C \sqcap D) = \{C', D', C' \sqcap D' | C' \in gqs(C), D' \in gqs(D)\} & \\
gqs(C \sqcup D) = \{C' \sqcup D' | C' \in gqs(C), D' \in gqs(D)\} & \\
gqs(\exists r.C) = \{\exists r.C' | C' \in gqs(C)\} & \\
gqs(\forall r.C) = \{\forall r.C' | C' \in gqs(C)\} & \\
gqs(\neg C) = \{\neg C' | C' \in sqs(C)\} & \\
sqs(C \sqcap D) = \{C' \sqcap D' | C' \in sqs(C), D' \in sqs(D)\} & \\
sqs(C \sqcup D) = \{C', D', C' \sqcup D' | C' \in sqs(C), D' \in sqs(D)\} & \\
sqs(\exists r.C) = \{\exists r.C' | C' \in sqs(C)\} & \\
sqs(\forall r.C) = \{\forall r.C' | C' \in sqs(C)\} & \\
sqs(\neg C) = \{\neg C' | C' \in gqs(C)\} &
\end{array}$$

As Schlobach himself notes, a simple consequence of this definition is that $\models C \sqsubseteq C'$ for every $C' \in gqs(C)$, and $\models D' \sqsubseteq D$ for each $D' \sqsubseteq sqs(D)$.

We slightly extend the base case of Definition 4 as follows:

**Definition 5 (Generalized and Specialized Qualified Subconcepts extended).** *We define $sqs(C)$ and $gqs(C)$ by adding to clauses in Definition 4 the following ones:*

$gqs(A) = \{A\} \cup \{gqs(D) \mid A \sqsubseteq D \in \mathbf{TBox}\}$ *if A is atomic*
$sqs(A) = \{A\} \cup \{sqs(C) \mid C \sqsubseteq A \in \mathbf{TBox}\}$ *if A is atomic*
$gqs(\neg A) = \{\neg A\} \cup \{gqs(D) \mid \neg A \sqsubseteq D \in \mathbf{TBox}\}$ *if A is atomic*
$sqs(\neg A) = \{\neg A\} \cup \{sqs(C) \mid C \sqsubseteq \neg A \in \mathbf{TBox}\}$ *if A is atomic*

Thus, we also take into account the axioms (i.e., concept inclusions) defined in a given **TBox**. This generalization allows us to move upward (by means of *gqs*), and downward (by means of *sqs*) in the hierarchy of concepts defined by a given **TBox** $\mathcal{T}$. Relying on the notions of *sqs* and *gqs*, we can define a partial ordering relation between concepts as follows:

**Definition 6.** *Let $C$ and $D$ be two concepts in a given* **TBox** $\mathcal{T}$*, we say that $C$ is more specific than $D$, denoted as $C \prec D$, iff at least one of the following relations holds: (i) $C \in sqs(D)$, or (ii) $D \in gqs(C)$.*

It is easy to see that $\prec$ is irreflexive, antisymmetric, and transitive; however, it is just partial because $\prec$ is not defined for any pair of concepts; i.e., there may exist two concepts $C$ and $D$ such that neither $C \prec D$ nor $D \prec C$ holds. As we will discuss later, the methodology we propose for determining which concepts represent properties to be made "typical" relies on the fact that concepts are considered in order from the most specific to the most general. In those situations where two concepts are not directly comparable with one another by means of $\prec$, we need some further tools. For this reason, we introduce the notions of *Concept Network* and *depth* of concepts. First of all, let $\mathcal{S}$ be the set of concepts (and subconcepts) occurring in a given **TBox**.

**Definition 7 (Concept Network).** *Given a TBox $\mathcal{T}$, a concept network for $\mathcal{T}$ is a graph $CN(\mathcal{T}) : \langle V, E\rangle$ where $V$ is a set of vertexes, each of which corresponds to a concept $C \in \mathcal{S}$, and $E$ is a set of oriented edges of the form $\langle C, D\rangle$, where $C$ and $D$ belong to $V$; an edge $\langle C, D\rangle$ is in $E$ iff $C \prec D$.*

**Definition 8 (Depth of a concept).** *Given a TBox $\mathcal{T}$, its corresponding concept network $CN(\mathcal{T})$, and a concept $C \in \mathcal{S}$, the depth of a concept $C \in \mathcal{T}$, denoted as $depth(C)$, corresponds to the length of the longest path from $\top$ to $C$.*

In principle, any two concepts $C$ and $D$ at the same depth can be considered by our methodology in either orderings $C \prec D$ or $D \prec C$. However, when both $C$ and $\neg C$ are at the same depth, we use a further criteria of preference based on the distance between concepts.

**Definition 9 (Distance between concepts).** *Given a TBox $\mathcal{T}$, its corresponding concept network $CN(\mathcal{T})$, and two concepts $C, D \in \mathcal{S}$, the distance between $C$ and $D$ in $\mathcal{T}$, denoted as $dist(C, D)$ is given by the shortest path in $CN(\mathcal{T})$ from $C$ to $D$, if $C \prec D$; $\infty$, otherwise.*

*Example 1.* Let us consider a simple example in which a **TBox** $\mathcal{T}_{student}$ contains the following concept definitions:
*Student* $\sqsubseteq \neg$ *TaxPayer*
*Student* $\sqcap$ *Worker* $\sqsubseteq$ *TaxPayer*

Of course, $\mathcal{T}_{student}$ is consistent only if there are no working students, and in the following we will show how it can be repaired by means of the typicality operator. For the time being, we are just interested in determining the depth of the concepts in $\mathcal{T}_{student}$. By applying Definition 6 we obtain:

$$\mathit{Student} \sqcap \mathit{Worker} \prec \mathit{Student} \prec \neg \mathit{TaxPayer};\ \mathit{Student} \sqcap \mathit{Worker} \prec \mathit{TaxPayer}.$$

It is easy to see that $\mathit{Student} \sqcap \mathit{Worker}$ is the deepest concept in our terminology. In fact, we have that both $\mathit{TaxPayer}$ and $\neg \mathit{TaxPayer}$ are at the same depth 1, as well as $\mathit{Worker}$; $\mathit{Student}$ is at depth 2; whereas $\mathit{Student} \sqcap \mathit{Worker}$ is at depth 3. However, since $dist(Student \sqcap Worker, TaxPayer) < dist(Student \sqcap Worker, \neg TaxPayer)$, we will prefer to consider $TaxPayer$ before $\neg TaxPayer$.

Definition 5 above does not take into account that the terminology may be inconsistent, and hence some of the concepts generated via the generalization (specialization) rules might be trivially contradictory (e.g., having the form $D \sqcap \neg D$). Moreover, the set of concepts within the $gqs$ ($sqs$) of a given concept $C$ are not necessarily consistent with one another. For instance, let us consider again $\mathcal{T}_{student}$; it is easy to see that $gqs(\mathit{Student} \sqcap \mathit{Worker})$ is the set:
$\{\mathit{Student} \sqcap \mathit{Worker}, \mathit{Student}, \mathit{TaxPayer}, \neg \mathit{TaxPayer}, \neg (\mathit{Student} \sqcap \mathit{Worker}), \mathit{Worker}, \neg \mathit{TaxPayer} \sqcap \mathit{Worker}, \neg (\mathit{Student} \sqcap \mathit{Worker}) \sqcap \mathit{Worker}, \mathit{TaxPayer}\}$.

Both $\mathit{TaxPayer}$ and $\neg \mathit{TaxPayer}$ are members of $gqs(\mathit{Student} \sqcap \mathit{Worker})$, but only one of them will be correct, the other will have to be pruned. As we have already mentioned above, $\mathit{TaxPayer}$ will be preferred as it is closer to $\mathit{Student} \sqcap \mathit{Worker}$. In the next section we propose a pruning technique that discards those concepts that, albeit generalize a concept $C$ (i.e., belong to $gqs(C)$), contradict the terminology $\mathcal{T}$. Our pruning technique relies on a compact representation of $gqs$. That is, rather than explicitly computing all concepts in $gqs(C)$ by recursively unfolding every $gqs(C')$ for each $C' \in gqs(C)$, we consider $gqs(C)$ as a list of concepts (not necessarily atomic), and $gqs$s.

For example, $gqs(\mathit{Student} \sqcap \mathit{Worker})$ can be compactly represented as the list of elements $\{\mathit{Student} \sqcap \mathit{Worker}, gqs(\mathit{Student}) \sqcap gqs(Worker), gqs(\mathit{Student}), gqs(Worker)\}$. Intuitively, $gqs(\mathit{Student} \sqcap \mathit{Worker})$ is given by the union of the concepts that are directly mentioned in the list (e.g., $\mathit{Student} \sqcap \mathit{Worker}$), or that belong to one of the $gqs$s mentioned in the list itself (e.g., $gqs(\mathit{Student})$).

Note that we keep the list associated with $gqs(C)$ ordered according to the depth (and distance when necessary) of the mentioned concepts. Also in this case, deepest concepts come first. However we have to pay some attention in dealing with $gqs$s. In fact, if $D \prec E$, then also $gqs(D) \prec E$, and hence $gqs(D) \prec gqs(E)$. However, if both $D$ and $gqs(D)$ are in $gqs(C)$, then $D \prec gqs(D)$. On the other hand, if $D$ and $F$ have the same depth, we put first the concept which is closer to $C$. In case, $D$ and $F$ are at the same distance from $C$ than their relative ordering is arbitrary. The idea is that $gqs(D)$ abstracts a set of concepts which are at least as specific as $D$, but possibly more general than $D$. Therefore, any concept more specific than $D$ is also more specific than $gqs(D)$. Vice versa, if $D \prec E$, then also $gqs(D) \prec E$ as $gqs(D)$ contains at least $D$.

## 4 Revising Terminologies Including Exceptions

In this section we present the methodology for automatically modify a **TBox** $\mathcal{T}$ whenever a detected inconsistency can be treated as an exception. The basic idea of our proposal is first to isolate a subset of $\mathcal{T}$ which is relevant for the inconsistency, and for this purpose we will adopt the notion of MUPS introduced in Definition 3. Then we reconsider all the concepts within a MUPS and try to identify which concepts describe characterizing properties to be made "typical". The intuition is that all the properties labeled as typical hold for all "normal" members of a given class $C$, but not necessarily for all members in $C$. That is, we admit that a subset of individuals in $C$ do not present the typical properties of the class, and hence are considered as exceptional.

Let us start by considering a coherent terminology $\mathcal{T}$, and assume we discover a new subsumption relation $newC \sqsubseteq D$ that once added to $\mathcal{T}$ introduces an incoherence. As mentioned above, the first step consists in restricting our attention to only those concepts that are relevant for the incoherence. These concepts are identified by $mups(\mathcal{T}, newC)$ [18]. For the sake of discussion, we will assume that $mups(\mathcal{T}, newC)$ will only contain a set of incoherent axioms, the approach can be easily extended to consider the more general situation in which $mups(\mathcal{T}, newC)$ is a set of sets.

In principle, we have to identify which concept subsumptions in $mups(\mathcal{T}, newC)$ are to be modified by introducing the **T** operator on the left-hand side of an axiom. The main issue in this process is that the terminology $\mathcal{T}$ might hide implicit knowledge (i.e., implicit subsumptions), and these implicit subsumptions are still hidden in $mups(\mathcal{T}, newC)$. Inferring implicit subsumptions is therefore essential in order to correctly identify the most specific properties to be made typical.

To reach this result, we propose a search process that is based on the following intuition: Given a concept $C$ in $mups(\mathcal{T}, newC)$, all the implicit subsumptions we are looking for are already contained in $gqs(C)$. In other words, for any concept $D \in gqs(C)$ (either atomic or not), the subsumption $C \sqsubseteq D$ must hold by definition of $gqs$. However, we have to be cautious when dealing with $gqs(C)$ as it potentially contains subsumptions which are irrelevant, or even incorrect, for the specific case under consideration. First of all, $gqs(C)$ considers all the possible concept definitions in $\mathcal{T}$, but we are just interested in those concepts mentioned within $mups(\mathcal{T}, newC)$. Thus, in considering the concepts mentioned within $gqs(C)$ we have to restrict ourselves only to those concepts which are also mentioned $mups(\mathcal{T}, newC)$; any other concept in $gqs(C)$ will be ignored as irrelevant. In addition, not all the generalizations suggested by $gqs(C)$ are consistent with the concepts in $mups(\mathcal{T}, newC)$. As we have already noted, $mups(\mathcal{T}, newC)$ might contain the subsumption relation $C \sqsubseteq D$; at the same time, the concept $\neg D$ appears in $gqs(C)$; implying that $C \sqsubseteq \neg D$. Of course, this second subsumption is in direct contrast with a subsumption in $mups(\mathcal{T}, newC)$; we say that $C \sqsubseteq \neg D$ *syntactically contradicts* $C \sqsubseteq D$. (Note that such a contradiction is just syntactic, and hence it can be checked by inspecting concepts in $mups(\mathcal{T}, newC)$.)

Another critical issue is that we are interested in finding the most specific properties that have to be considered as typical. This is also the reason why we look for implicit subsumptions: we want to discover what properties are inherited by the concepts in $mups(\mathcal{T}, newC)$. Thus, when we look for inclusions, we have to proceed from the most specific concepts in $mups(\mathcal{T}, newC)$ to the most general ones. In this way, we can progressively build a new set of subsumption relations $\mathcal{S}$, in which the typicality operator is added to only those specific properties that are relevant for dealing with the exception introduced by $newC$.

### 4.1 Algorithms

In this section we give a detailed description of the methodology we propose for automatically managing, by means of the typicality operator, a concept $newC$ with exceptional properties $D$. We give for granted that $mups(\mathcal{T}, newC)$ has already been computed by means of one of the calculi proposed in [8]. Thus, $mups(\mathcal{T}, newC)$ identifies a minimal subset of concept inclusions that are relevant for the inconsistency arising when $\mathcal{T}$ is extended with $newC \sqsubseteq D$.

**Algorithm 1** FindSubsumptionsMain($mups(\mathcal{T}, newC)$)

```
1: $\mathcal{S} \leftarrow \emptyset$
2: for all depth $l$ of concepts in $mups(\mathcal{T}, newC)$, from the highest to the lowest do
3:     for all $C_i$ in $mups(\mathcal{T}, newC)$ such that $depth(C_i) = l$ do
4:         $\mathcal{S}_i \leftarrow$ FindSubsmption($C_i$, $gqs(C_i)$, $\mathcal{S}$)
5:     end for
6:     if $\mathcal{S} \cup \bigcup \mathcal{S}_i$ is coherent then
7:         $\mathcal{S} \leftarrow \mathcal{S} \cup \bigcup \mathcal{S}_i$
8:     else
9:         for all $\mathcal{S}_i$ do
10:            for all $C \sqsubseteq D \in \mathcal{S}_i$, such that $C$ does not occur in $D$ do
11:                $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{T}(C) \sqsubseteq D\}$
12:            end for
13:        end for
14:    end if
15: end for
16: return $S$
```

**FindSubsumptionsMain** To restore the consistency in $\mathcal{T}$, we propose to rewrite the inclusion relations in $mups(\mathcal{T}, newC)$ by characterizing some of them with the typicality operator. As said above, we reach this goal by considering all concepts mentioned in $mups(\mathcal{T}, newC)$ in depth ordering, starting from the deepest ones up to the most generic concepts. Algorithm 1 simply shows this loop over the concepts in $mups(\mathcal{T}, newC)$. At each iteration, a set $\mathcal{S}$ of inclusions is incrementally extended with the new relations discovered by function *FindSubsumptions*. In case several concepts $C_i$ have the same depth, a further consistency check among their corresponding sets $\mathcal{S}_i$s is performed to deal with inconsistent properties, coming from different $\mathcal{S}_i$s, and inherited by a given concept $F$. For instance, let us assume that $F \sqsubseteq C \sqcap D \in \mathcal{S}$, $C$ and $D$ have the same depth and distance from $F$, thus we cannot prefer one to the other. In addition, by invoking *FindSubsumptions* over $C$ and $D$ we get, respectively, $C \sqsubseteq E \in \mathcal{S}_C$

and $D \sqsubseteq \neg E \in \mathcal{S}_D$. Since $F$ inherits both $E$ and $\neg E$, we do not conclude anything about $F$ having, or not, property $E$; however, to build a coherent $\mathcal{S}$, $\mathbf{T}(C) \sqsubseteq E$ and $\mathbf{T}(D) \sqsubseteq \neg E$ are added to $\mathcal{S}$.

**FindSubsumptions** Algorithm 2 sketches the main steps of function *Find-Subsumption*: it takes as inputs a concept $C_i$ mentioned in $mups(\mathcal{T}, newC)$, the corresponding $gqs(C_i)$ given as a list in compact form, and the set $\mathcal{S}$ of inclusions found so far. The intuition is that all implicit inclusions we are looking for have the form $C_i \sqsubseteq D$, where $D \in gqs(C_i)$.

First of all, the algorithm initializes some local structures: $\mathcal{S}_i$ will contain the subsumptions discovered in this invocation, *BlackList* will only contain the *gqs* of concepts that have been pruned: future occurrences of concepts belonging to these *gqs* have to be discarded; *Queue* will either contain concept nodes or complex nodes: a concept node represents a concept (not necessarily atomic), whereas a complex node represents a structure where at least a *gqs* is mentioned (i.e., a complex node is an abstraction of a number of concepts). All nodes in *Queue* are kept ordered from the most specific to the most general according to the depth of the concepts they contain. At the beginning of the algorithm (see lines 3- 12), *Queue* is initialized with the elements in $gqs(C)$.

After these preliminary steps, the algorithm loops over the elements in *Queue*. At each iteration, the first node $n$ is removed from the head of *Queue*. The algorithm then checks whether $n$ is a concept node or a complex node.

*Handling concept nodes* (lines 15 - 28). If $n$ is a concept node, for instance representing concept $D$, the algorithm has discovered a possible inclusion $C_i \sqsubseteq D$ to be added to $\mathcal{S}_i$. However, if $C_i \sqsubseteq \neg D$ is already in $\mathcal{S} \cup \mathcal{S}_i$, relation $C_i \sqsubseteq D$ has to be discarded. Moreover, since we know that $D$ is not acceptable, then any other concept which generalizes $D$ is not acceptable, too; thereby we remove from *Queue* any node $e$ belonging to $gqs(D)$ (line 18), and then add $gqs(D)$ in *BlackList* (line 19). Otherwise ($C_i \sqsubseteq \neg D$ is not in $\mathcal{S} \cup \mathcal{S}_i$), $C_i \sqsubseteq D$ represents a new piece of knowledge that we can add to $\mathcal{S}_i$. If $\mathcal{S} \cup \mathcal{S}_i \cup \{C_i \sqsubseteq D\}$ is coherent, the new relation is added to $\mathcal{S}_i$ as it is (line 22). Conversely, we have discovered one of the most specific properties of $C_i$ that have to be made typical. That is, in order to restore the consistency in $\mathcal{T}$ when we also consider the exceptional properties of *newC*, the inclusion $C_i \sqsubseteq D$ has to be rewritten as $\mathbf{T}(C_i) \sqsubseteq D$ (line 24).

Note that, since we add either $C_i \sqsubseteq D$ or $\mathbf{T}(C_i) \sqsubseteq D$, we can conclude that any other concept in $gqs(\neg D)$ will not be acceptable as it would introduce an inconsistency. Thus, we can remove from *Queue* any node $e$ which is a generalization of $\neg D$ (line 26), and then add $gqs(\neg D)$ in *BlackList* (line 27).

*Handling complex nodes* (lines 29 - 32). In case the head node $n$ is a complex node, then it must mention at least a *gqs*, which abstracts a number of concepts. Dealing with a complex node means generating a number of child-nodes by expanding one of the *gqs* mentioned in $n$. The *ExpandComplexNode* function is shown in Algorithm 3. The set of new nodes $NewNodes$ returned by *ExpandComplexNode* are enqueue in *Queue* in the specificity ordering.

**Algorithm 2** FindSubsumptions($C_i, gqs(C_i), \mathcal{S}$)

```
 1: $\mathcal{S}_i \leftarrow \emptyset$
 2: $BlackList \leftarrow \emptyset$
 3: $Queue \leftarrow \emptyset$
 4: for all element $e \in gqs(C_i)$, in the specificity ordering do
 5:     if $e$ is a concept $E$ then
 6:         create a concept node $n[E]$
 7:         enqueue $n[E]$ in $Queue$ in the ordering
 8:     else                    ▷ $e$ is a generalization of a concept $E$; i.e., $gqs(E)$
 9:         create a complex node $n[gqs(E)]$
10:         enqueue $n[gqs(E)]$ in $Queue$ in the ordering
11:     end if
12: end for
13: while $Queue$ is not empty do
14:     $n \leftarrow Head(Queue)$
15:     if $n$ is a concept node then
16:         Let $D$ be the concept in $n$; then $\mathcal{S}_i$ is possibly extended to include $C_i \sqsubseteq D$
17:         if $C_i \sqsubseteq \neg D \in \mathcal{S} \cup \mathcal{S}_i$ then
18:             remove from $Queue$ any node $e$ such that $e \in gqs(D)$
19:             enqueue $gqs(D)$ in $BlackList$
20:         else                                        ▷ $C_i \sqsubseteq \neg D \notin \mathcal{S}$
21:             if $\mathcal{S} \cup \mathcal{S}_i \cup \{C_i \sqsubseteq D\}$ is coherent then
22:                 $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{C_i \sqsubseteq D\}$
23:             else
24:                 $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{\mathbf{T}(C_i) \sqsubseteq D\}$
25:             end if
26:             remove from $Queue$ any node $e$ such that $e \in gqs(\neg D)$
27:             enqueue $gqs(\neg D)$ in $BlackList$
28:         end if
29:     else                                            ▷ $n$ is a complex node
30:         $NewNodes \leftarrow$ ExpandComplexNode($n$, $BlackList$)
31:         enqueue each node $n' \in NewNodes$ in $Queue$ in the ordering
32:     end if
33: end while
34: return $\mathcal{S}_i$
```

**Algorithm 3** ExpandComplexNode($n$, $BlackList$)

```
 1: $List \leftarrow \emptyset$
 2: Let $gqs(D)$ be one of the most specific $gqs$ mentioned in $n$
 3: for all element $e \in gqs(D)$ such that $e \notin gqs(F)$, $\forall gqs(F) \in BlackList$ do
 4:     create a node $n'$ by substituting $gqs(D)$ with $e$ in $n$
 5:     if $n'$ is an inconsistent concept then
 6:         discard $n'$
 7:     else
 8:         put $n'$ in $List$
 9:     end if
10: end for
11: return $List$
```

**ExpandComplexNode** The expansion of a complex node is outlined in Algorithm 3. It takes as inputs the complex node $n$ that has to be expanded, and the *BlackList* containing the *gqs*s of those concepts that have been pruned off so far. The algorithm returns a list of new nodes, either concept or complex.
After having initialized *List*, (line 1), the algorithm selects one of the most specific *gqs* in $n$. In fact, since $n$ is a complex node, then it must mention at least one *gqs* (line 2). Let us assume that $gqs(D)$ has been selected, remember that $gqs(D)$ is compactly encoded as a list of concepts or *gqs*s. The algorithm therefore loops over the elements $e$ in the list $gqs(D)$, if $e$ belongs to any $gqs(F) \in$ *BlackList*, then $e$ can be pruned off as it has already been determined that $F$, and then any other concept in $gqs(F)$, is not consistent with some axioms in $\mathcal{S}$ (line 3). Otherwise, $e$ can be used to create a new node $n'$. More precisely, $n'$ is obtained by substituting $gqs(D)$ with $e$ in $n$. Of course, $n'$ can either be a concept node or a complex node (line 4). As noted above, the generalizations of a given concept $D$ might contain contradictory concepts since the initial set of axioms in $mups(\mathcal{T}, newC)$ is inconsistent. Thus, before adding $n'$ to *List*, we first check whether $n'$ is contradictory, in which case we discard it (line 5). Otherwise, $n'$ is added to *List* (line 8). The algorithm terminates by returning the, possibly empty, list of new nodes (line 11).

Let $newC \sqsubseteq D$ be the subsumption relation that, once added to a **TBox** $\mathcal{T}$, generates an inconsistency, and let $mups(\mathcal{T}, newC)$ the minimal subset of axioms in $\mathcal{T}$ preserving the inconsistency, then we can prove the following properties.

**Proposition 1.** *The set $\mathcal{S}$ of concept inclusions generated by Algorithm 1 with the invocation* FindSubsumptionsMain($mups(\mathcal{T}, newC)$) *is coherent.*

Intuitively, $\mathcal{S}$ is only modified either in line 7 or in line 11 of algorithm *FindSubsumptionsMain.* In the first case, we already know that all the sets $S_i$s inferred for all concepts at a given depth $l$ are altogether coherent. In the second case, instead, we know that an inconsistency has been detected; thereby we "correct" the axioms in $S_i$s by means of $\mathbf{T}$, yielding the consistency of $\mathcal{S}$. In fact, for the properties $\mathcal{ALC}^{\mathbf{R}}_{min}\mathbf{T}$ in [7], since $\mathbf{T}$ is nonmonotonic, we have that $\mathcal{S}$ is coherent.

**Proposition 2.** *Let $\mathcal{S}$ be the set of concept inclusions generated by Algorithm 1, then either $newC \sqsubseteq D$ or $\mathbf{T}(newC) \sqsubseteq D$ belongs to $\mathcal{S}$.*

This follows directly by the fact that $newC \sqsubseteq D$ has generated an inconsistency in $\mathcal{T}$, and then $mups(\mathcal{T}, newC)$ necessarily includes such a relation. Algorithm 1 simply reconsiders all the axioms in $mups(\mathcal{T}, newC)$, and builds a new set $\mathcal{S}$ of axioms which contains, at least, the same axioms as in $mups(\mathcal{T}, newC)$, but at least one has been modified by the typicality operator.

**Theorem 1.** *Let $\mathcal{T}'$ be a new terminology obtained as $T' = \mathcal{T} \setminus mups(\mathcal{T}, newC) \cup \mathcal{S}$. The new terminology $\mathcal{T}'$ is coherent.*

This allows us to conclude that once we have computed $\mathcal{S}$, we have a means for correcting the initially incoherent terminology $\mathcal{T}$. The simplest way to do that is to substitute the axioms in $mups(\mathcal{T}, newC)$ with $\mathcal{S}$. Further details and proofs are omitted due to space limitations.

*Example 2.* Let us consider again the terminology $\mathcal{T}_{student}$ presented in Example 1. Algorithm 1 considers the concepts in the ordering: $Student \sqcap Worker$, $Student$, $TaxPayer$, $\neg TaxPayer$, $Worker$. The first invocation of *FindSubsumptons* (Algorithm 2) is therefore over the inputs: $Student \sqcap Worker$, $gqs(Student \sqcap Worker)$, and $\emptyset$ (i.e., at the beginning $\mathcal{S}$ is empty). The result of such a first invocation is the set $\mathcal{S} = \{Student \sqcap Worker \sqsubseteq Student; Student \sqcap Worker \sqsubseteq TaxPayer; Student \sqcap Worker \sqsubseteq Worker\}$. The second invocation of *FindSubsumptions* is over the inputs: $Student$, $gqs(Student)$, $\mathcal{S}$. In this case, the algorithm will find the subsumption $Student \sqsubseteq \neg TaxPayer$, which is incoherent with the subsumptions already in $\mathcal{S}$, thus, $\neg TaxPayer$ is considered as the typical property of $Student$; in other words, $\mathcal{S}$ is modified by adding $\mathbf{T}(Student) \sqsubseteq \neg TaxPayer$. It is easy to see that no further subsumptions are added in $\mathcal{S}$ when *FindSubsumptions* is invoked over the remaining concepts $TaxPayer$, $Worker$, and $\neg TaxPayer$. In conclusion, the original terminology $\mathcal{T}_{student}$, can be rewritten as $\mathcal{T}'_{student}$:

$$\begin{array}{lr} Student \sqcap Worker \sqsubseteq Student & Student \sqcap Worker \sqsubseteq TaxPayer \\ Student \sqcap Worker \sqsubseteq Worker & \mathbf{T}(Student) \sqsubseteq \neg TaxPayer \end{array}$$

By the properties of the DL $\mathcal{ALC}^{\mathbf{R}}_{min}\mathbf{T}$, from the resulting knowledge base above we are able to reason about defeasible inheritance. For instance, if we have $Student(franco)$ (Franco is a student), we are able to infer $\neg TaxPayer(franco)$ (Franco does not pay taxes), however this conclusion is retracted if we further discover that $Worker(franco)$ (Franco is also a worker), from which we obtain $TaxPayer(franco)$. We are also able to deal with *irrelevance*: for instance, $\mathbf{T}(Student \sqcap Fat) \sqsubseteq \neg TaxPayer$ can be inferred, and the above conclusions still hold even if we further know $Fat(franco)$ (Franco is fat).

*Example 3.* Let us consider a simplified variant of the Pizza ontology distributed together with Protégé. In particular, let us assume that a **TBox** $\mathcal{T}_{tops}$ contains the following subsumption relations:

$ax_1$ : $FatToppings \sqsubseteq PizzaToppings$,
$ax_2$ : $CheesyToppings \sqsubseteq FatToppings$,
$ax_3$ : $VegetarianToppings \sqsubseteq \neg FatToppings$.

Now, let us assume that we discover that there also exists the *tofu* cheese, which is made of curdled soybeanmilk. Thus, we change $\mathcal{T}_{tops}$ by adding the following:

$ax_4$ : $CheesyVegetarianToppings \sqsubseteq CheesyToppings \sqcap VegetarianToppings.$

Let the **ABox** $\mathcal{A}_{tops}$ contain $CheesyVegetarianToppings(tofu)$. The resulting knowledge base is inconsistent, respectively, the knowledge base obtained by adding $CheesyVegetarianToppings(tofu)$ to the initial ABox is inconsistent, since $FatToppings$ and $DieteticToppings$ are disjunct concepts, whereas $CheesyVegetarianToppings$ falls in their intersection.

The tofu cheese is an exception, and the first step to tackle is to compute $mups(\mathcal{T}_{tops},\ \mathit{CheesyVegetarianToppings}) = \{ax_2, ax_3, ax_4\}$. Concept inclusions (even implicit) are taken into account: possibly, some of them will be "corrected" by means of the typicality operator in order to accommodate the exceptional *CheesyVegetarianToppings* concept. In particular, the inclusions discovered are:

$\mathcal{S}_{tops} = \{$
$\mathit{CheesyVegetarianToppings} \sqsubseteq \mathit{CheesyToppings}$,
$\mathit{CheesyVegetarianToppings} \sqsubseteq \mathit{VegetarianToppings}$,
$\mathbf{T}(\mathit{CheesyToppings}) \sqsubseteq \mathit{FatToppings}$,
$\mathbf{T}(\mathit{VegetarianToppings}) \sqsubseteq \neg \mathit{FatToppings}$
$\}$

The concept inclusions in $mups(\mathcal{T}_{tops}, \mathit{CheesyVegetarianToppings})$ can be now substituted in $\mathcal{T}_{tops}$ with the set $\mathcal{S}_{tops}$. Theorem 1 ensures that the new $\mathcal{T}_{top}$ so obtained is now coherent and correctly addresses the exceptional concept *CheesyVegetarianToppings*. By the properties of $\mathcal{ALC}^{\mathbf{R}}_{min}\mathbf{T}$, from the resulting knowledge base, we will be able to reason about defeasible inheritance: for instance, if we know $\mathit{CheesyToppings}(\mathit{brie})$ (Brie is a cheesy topping), we conclude that $\mathit{FatToppings}(\mathit{brie})$ (Brie is a fat topping), whereas for tofu we say nothing about being a fat topping or not. We are also able to deal with *irrelevance*: for instance, it can be inferred that, normally, cheesy potatoes toppings are fat toppings, i.e. $\mathbf{T}(\mathit{Cheesy} \sqcap \mathit{Potatoes}) \sqsubseteq \mathit{FatToppings}$.

## 5 Conclusions and Future Issues

We have presented some preliminary results in defining a methodology for automated revision of a DL terminology in presence of exceptions. We exploit techniques and algorithms proposed in [8], which are also extended to more expressive DLs such as $\mathcal{SHOIN}$, corresponding to ontology language OWL-DL. On the one hand, we aim at extending our approach to such expressive DLs. On the other hand, we intend to develop an implementation of he proposed algorithms, by considering the integration with existing tools for manually modifying ontologies when inconsistencies are detected.

## Acknowledgements

The authors are supported by the twin projects ODIATI#1 and ODIATI#2: "Ontologie, DIAgnosi e TIpicalità nelle logiche descrittive" of the local research funds 2013 by the Università degli Studi di Torino - part B, supporting young researchers.

## References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook - Theory, Implementation, and Applications, 2nd edition. Cambridge (2007)
2. Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. Autom. Reasoning 14(1), 149–180 (1995)
3. Casini, G., Straccia, U.: Rational Closure for Defeasible Description Logics. In: Janhunen, T., Niemelä, I. (eds.) Proceedings of the 12th European Conference on Logics in Artificial Intelligence (JELIA 2010). Lecture Notes in Artificial Intelligence, vol. 6341, pp. 77–90. Springer, Helsinki, Finland (September 2010)
4. Casini, G., Straccia, U.: Defeasible Inheritance-Based Description Logics. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 813–818. Morgan Kaufmann, Barcelona, Spain (July 2011)
5. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: ALC+T: a preferential extension of Description Logics. Fundamenta Informaticae 96, 1–32 (2009)
6. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A NonMonotonic Description Logic for Reasoning About Typicality. Artificial Intelligence 195, 165–202 (2013)
7. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Minimal Model Semantics and Rational Closure in Description Logics . In: Eiter, T., Glim, B., Kazakov, Y., Krtzsch, M. (eds.) Informal Proceedings of the 26th International Workshop on Description Logics (DL 2013). CEUR Workshop Proceedings, vol. 1014, pp. 168 – 180. Ulm, Germany (7 2013)
8. Kalyanpur, A., Parsia, B., Cuenca-Grau, B., Sirin, E.: Axiom pinpointing: Finding (precise) justifications for arbitrary entailments in $\mathcal{SHOIN}$ (owl-dl). Technical report, UMIACS, 2005-66 (2006)
9. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C.: Repairing unsatisfiable concepts in owl ontologies. In: ESWC. pp. 170–184 (2006)
10. Ke, P., Sattler, U.: Next Steps for Description Logics of Minimal Knowledge and Negation as Failure. In: Baader, F., Lutz, C., Motik, B. (eds.) Proceedings of Description Logics. CEUR Workshop Proceedings, vol. 353. CEUR-WS.org, Dresden, Germany (May 2008)
11. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
12. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)
13. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging owl ontologies. In: WWW. pp. 633–640 (2005)
14. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32 (1), 57–96 (1987)
15. Schlobach, S.: Diagnosing terminologies. In: AAAI. pp. 670–675 (2005)
16. Schlobach, S., Cornet, R.: Explanation of terminological reasoning: A preliminary report. In: Description Logics (2003)
17. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: IJCAI. pp. 355–362 (2003)
18. Schlobach, S., Huang, Z., Cornet, R., van Harmelen, F.: Debugging incoherent terminologies. Journal of Automated Reasoning 39(3), 317–349 (2007)