View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositório da Universidade Nova de Lisboa

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Derivation Methods for Hybrid Knowledge Bases with Rules and Ontologies

Ana Sofia Gomes (aluna nº 26335)

2º Semestre de 2008/09

Julho de 2009

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Derivation Methods for Hybrid Knowledge Bases with Rules and Ontologies

Ana Sofia Gomes (aluna nº 26335)

Orientador: Prof. Doutor José Júlio Alferes

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2008/09

Julho de 2009

Acknowledgements

First of all, I would like to thank my advisor, José Júlio Alferes, for his incredible support. Right from the start, during the first semester of this work, when we were 2700 km apart and meeting regularly via Skype, until the end of this dissertation, he was always committed and available for discussions, even when he had lots of other urgent things to do.

A really special thanks to Terrance Swift, whom acted as an advisor, helping me a lot in the second implementation, and correcting all XSB's and CDF's bugs. This implementation wouldn't surely have reached such a fruitful end without his support.

I would also like to thank all my colleagues and friends at FCT for the great work environment and for not letting me take myself too serious. A special thanks to my colleagues from Dresden for encouraging me to work even when there were so many other interesting things to do as an Erasmus student.

I'm indebted to Luís Leal, Bárbara Soares, Jorge Soares and Cecília Calado, who kindly accepted to read a preliminary version of this report and gave me their valuable comments.

For giving me working conditions and a partial financial support, I acknowledge the Departamento de Informática of the Faculdade de Ciências e Tecnologias of Universidade Nova de Lisboa.

Last, but definitely not least, I would like to thank my parents and all my family for their continuous encouragement and motivation. A special thanks to Bruno for his love, support and patience.

Abstract

The raising popularity of the Semantic Web to describe concepts of the applications domains has revealed some aspects that could not be dealt by the Web Ontology Language (OWL) but are easily expressible in rule languages.

These limitations of OWL made clear the need of integrating nonmonotonic rule-based systems with monotonic ontologies in order to achieve the complete realisation of Semantic Web. However, this integration cannot be done straightforwardly as it faces many semantic and computational problems. These factors have sparked a heated debate which led to a research in different directions.

Moreover, the combination of such two prominent subjects is not only useful in the Semantic Web context. In fact, there are many use case applications, particularly applied to a biomedical domain, where such interoperation between rules and ontologies is desirable.

In the first part of this thesis we present a structured overview of the current state of the art on the subject of combining rules and ontologies. The objective is to describe several approaches that have appeared on the last years, explaining how they achieve their different degrees of integration.

We then focus on one specific proposal for the combination of rules and ontologies – MKNF Well-Founded Semantics [34] – that possesses some desirable decidability and complexity properties.

To finish, we conclude with two prototype implementations for such approach – a first bottom-up implementation based on the definition of the semantics; and a goal-driven implementation that, by being query-oriented, reduces the computational process to the set of atoms of which the given query depend on. For these implementations we use XSB along with CDF, an ontology management system complete for $\mathcal{ALCQ}$ extended with relational hierarchies and product classes. This represents an important contribution as it may be the first implementations for hybrid knowledge bases that combine nonmonotonic rules and DL ontologies, where the combination is such that both the rules can refer to predicates in the ontology, and the ontology can refer to predicates defined in the rules.

Keywords: Hybrid knowledge bases, Semantic Web, ontologies, description logics, rules, logic programming, well-founded semantics, nonmonotonic reasoning.

Resumo

A crescente popularidade da Web Semântica, para descrever conceitos de domínios aplicacionais, revelou diversos aspectos que não são suportados pela *Web Ontology Language* (OWL), mas que são facilmente exprimíveis em linguagens de regras comuns.

Estas limitações do OWL tornaram clara a necessidade de integrar sistemas não-monotónicos de regras com ontologias monotónicas, por forma a permitir alcançar todos os objectivos da Web Semântica. No entanto, esta integração não pode ser efectuada directamente, pois enfrenta vários problemas tanto computacionais como semânticos. Estes factores desencadearam um debate aceso na comunidade, o que levou a que a investigação neste tema tenha tomado diversas direcções.

Por outro lado, a combinação destes dois componentes não é apenas útil no contexto da Web Semântica, uma vez que começam a surgir diversos de casos de uso – particularmente na área de biomédica – onde esta interligação de regras com ontologias é desejável.

Na primeira parte deste documento apresentamos um resumo estruturado sobre o estado actual das diferentes investigações para conciliar regras e ontologias. O objectivo desta primeira parte é discutir as várias abordagens que surgiram nos últimos anos, descrevendo como são alcançados os diferentes níveis de integração.

Posteriormente, enfatizamos uma das propostas mais sólidas até ao momento para combinar regras com ontologias – *MKNF well-founded semantics* [34] – que possui propriedades de decisão e complexidade desejáveis.

Para finalizar, apresentamos duas diferentes implementações para esta abordagem – uma primeira implementação *bottom-up* baseada na definição da semântica considerada; e uma implementação *goal-driven* que, por ser orientada para responder a uma determinada *query* reduz o processo de computação ao conjunto de literais relevantes para a inferência da mesma. Para estas implementações escolhemos utilizar o XSB em conjunto com um sistema de gestão de ontologias – CDF – completo para a lógica de descrição $\mathcal{ALCQ}$, estendida com hierarquias relacionais e classes de produto. Estes dois protótipos são uma importante contribuição na medida em que representam as primeiras implementações para bases de conhecimento capazes combinar regras não-monótonicas com ontologias baseadas em lógicas de descrição, em que esta combinação permite que as regras referenciem a predicados da ontologia e vice-versa.

Palavras-chave: Bases de conhecimento híbridas, Web Semântica, lógicas de descrição, regras, programação em lógica, semântica bem-fundada, raciocínio não-monotónico.

Contents

Acronyms

CDF Coherent Description Framework

CWA Closed World Assumption

DL Description Logic

FOL First Order Logic

KR Knowledge Representation

LP Logic Programming

MKNF Minimal Knowledge and Negation as Failure

OWA Open World Assumption

OWL Web Ontology Language

PSM Partial Stable Model

RIF Rule Interchange Format

SNA Standard Name Assumption

SWRL Semantic Web Rule Language

UNA Unique Name Assumption

W3C World Wide Web Consortium

WFM Well-Founded Model

WFS Well-Founded Semantics

1. Introduction

1.1 Context – Combining Ontologies and Rules

The combination of rules and ontologies has become a hot topic in the ongoing development of the Semantic Web, as the need for a more powerful formalism arises [27, 28].

The growing popularity of the Semantic Web to describe concepts of the application domains has revealed some aspects that could not be dealt by the OWL, but are easily expressible in rule languages. In fact, there are several important modelling problems that are hard (if not impossible) to solve using OWL alone, but are smoothly addressed by rule languages, such as:

1. **Relation Expressivity.** OWL can only express axioms that can be structured in a tree-like way. Yet, there are many applications that require modelling general relational structures. For example, the following rule:

   $$hasAunt(x, y) \leftarrow hasParent(x, z), hasSibling(z, y), Female(y)$$

   destroys the tree model property and thus cannot be modelled in OWL, as it requires a triangle to express the relationship between the person, the parent and the aunt [44].

2. **Polyadic Predicates.** In OWL it is only possible to express unary and binary predicates. However, there are several situations where it would be desirable to define predicates of arbitrary arity.

3. **Integrity Constraints.** In OWL, domain and range restrictions constrain the type of objects that can be related by a role. For example, suppose that we want to state that "a social security number must be known for each person":

   $$Person \sqsubseteq \exists hasSSN.SSN$$

   However, adding a person without a social security number to the ontology will lead to the inference that the person in question has some unknown social security number, which is not the behaviour expected.

4. **Modelling Exceptions.** One of the main drawbacks of OWL is its inability to express constraints and exceptions. For example, suppose we want to model that most people have the heart on the left, but some people (called dextrocardiacs) have it on the right side of the body [42]. This exception cannot be modelled in OWL, as:

   $$\begin{aligned} Human &\sqsubseteq HeartOnLeft \\ Dextrocardiac &\sqsubseteq Human \\ Dextrocardiac &\sqsubseteq \neg HeartOnLeft \end{aligned}$$

   make the concept $Dextrocardiac$ unsatisfiable.

To express all of these concepts, it is necessary to go beyond first-order logic (FOL) and employ some kind of nonmonotonic formalism. Logic Programming (LP) is seen as a way to overcome several shortcomings of OWL, providing the necessary tools to model constrains and exceptions over Description Logic (DL) knowledge bases.

The other motivation for the combination of rules and ontologies is to provide OWL with the possibility to employ Closed World Reasoning.

LP and OWL are based on two opposite paradigms. A logic program is seen as a description of a single world, over which knowledge is complete. This way, LP adheres to Reiter's Closed World Assumption (CWA) [46], where incomplete knowledge about a proposition is resolved by turning it into falsity. Formally, if a theory $\mathcal{T}$ does not logically entail a ground atom A, then its negation $not\ A$ is concluded. On the other hand, ontologies, based on FOL sentences, rest in the Open World Assumption (OWA). Here, conclusions about propositions which can not be proven to be true in all possible worlds are kept open, and incomplete information is treated agnostically. Under a theory $\mathcal{T}$ it might be that neither $\mathcal{T} \models A$ nor $\mathcal{T} \models \neg A$ holds for a proposition A.

OWA is often reasonable in the Semantic Web context, where we have a wide set of knowledge sources. Yet, in situations where one has complete knowledge over a given source, it would be desirable to "turn on" the CWA, in order to attain the benefits of nonmonotonicity, but without giving up OWL's open world semantics in general.

As an illustration, consider Example 1.1 following next.

Example 1.1. *Consider a scenario application for a Customs House where an ontology is used to model and classify the aspects of imports and exports. An ontology with such characteristics would embrace several hundreds of axioms. In these axioms, as an example, one can define whether a shipment should be considered suspicious based on its country of origin. Intuitively, we could think that a Scandinavian country is considered as a safe country* 1.1*, as well as state Norway as a Scandinavian country, which would allow us to infer Norway as a safe country. Moreover, shipments from the government should also not be inspected.*

$$\begin{array}{l} ScandinavianCountry \sqsubseteq SafeCountry \quad ScandinavianCountry(Norway) \\ GovShipment \sqsubseteq \neg Inspect \end{array} \tag{1.1}$$

Still, it would be desirable to derive what actions to perform when receiving shipments. To this end, the need to define rules over the ontology's knowledge base comes arise in order to cope with incomplete information and to specify priorities between actions. For instance, one would like to define that a shipment from a not safe country should be inspected, which can be done by using default negation from rules 1.2*.*

$$inspect(x) \leftarrow hasShipment(x, country), \mathbf{not}\ SafeCountry(country). \tag{1.2}$$

However, even for countries considered as safe, a shipment should be inspected when their content is abnormal regarding its origin. As an example, a shipment containing tropical fruit

should be considered suspect if it is from a Scandinavian country. To express such "normality" concept, it emerges the need to employ nonmonontonic formalisms in a way that what is normal is stated directly in the knowledge base, and abnormality appears as the default negation of normal.

In this sense, we recur to closed world assumption, considering that everything behaviour that could not be inferred as normal should be seen as abnormal using default negation 1.3.

$$inspect(x) \leftarrow hasShipment(x, country), \mathbf{not}\ normal(x, country). \tag{1.3}$$

Yet, if otherwise we employ open world assumption when some behaviour A is not expected in the program, we wouldn't be able to derive neither that A is normal nor that is abnormal and consequently, it would not be possible to infer what action to perform, since our information for A is incomplete.

Hybrid knowledge bases are defined as the union between nonmonotonic Logic Programming with monotonic ontologies.

The combination of the two seeks to reap the best of both formalisms – the ability of LP to express complex rules and the power of DLs to structure knowledge in terms of concepts and relations.

However, achieving a conceptually clear integration of DLs and rules is not an easy task. Since DLs are based on a tractable fragment of FOL, which is monotonic, and LP rules are nonmonotonic, their semantics differs considerably, resulting in two paradigms which seem to be fundamentally incompatible. Consequently, achieving a meaningful and intuitive combined semantics is not straightforward. Moreover, the decidability issue is solved in each of DLs and LP in a quite different way: in DLs decidability is achieved by restricting the form of possible formulas (as those restriction pointed out in points 1-3 above), whilst still allowing potentially infinite domains and first order quantifiers; whereas in LP decidability is accomplished by restricting to finite domains. Given these differences, achieving an overall decidable and tractable combined semantics represents a difficult challenge. These factors led to research on the topic of ontologies and rules that has spawned into several different directions in very recent years, and with numerous novel proposals for such a combined semantics.

The existing proposals for combined knowledge bases including DL ontologies and rules can be divided into two main categories.

In a first category of proposals, the main concern is with points 1-3 above, while still keeping decidability and, if appropriate, also tractability. I.e. they are concerned to allow some special types of formulas that are usually possible in rules, connected together with constructs typical from DLs. Normally they restrict some constructs to preserve decidability, and to avoid raising (too much) the computational complexity. All these proposal, not addressing point 4, are monotonic, and conforming to OWA. As such, since in the end all can be translated to FOL, the issue of coming up with an intuitive combined semantics is not important for these approaches

while those of decidability, tractability, and also accompanying implementations are the points to solve.

The second category of proposals includes those that also aim at dealing with point 4 above, and truly combine nonmonotonic rules with monotonic descriptions of ontologies. In the existing proposals in this category, the main concern up to now has been to come up with an appropriate semantics for the combination and, to possibly simplify the decidability issue, by constraining more the available constructs. Also, not surprisingly, not much work exists to this date on tractability issues, proof procedures, and implementations. In fact, to the best of our knowledge, no implementation exists for such proposals, except for those that, though in this category, make strong syntactic restrictions on the way the combination is done. For example such proposals, that we call "Hybrid integration", may allow rules to make use of predicates defined by the DL ontology, but not allow the DL ontology to refer to predicates defined in the rules. On the other hand, the so called "Homogeneous integration" proposals allow both rules to make use of predicates in the ontology, and the ontology to refer to predicates defined in the rules.

1.2 Proposed work and contributions

In this dissertation we start by presenting a structured overview of the current state of the art on the subject of combining rules and ontologies. This, in itself, is already a contribution, since such a comprehensive state of the art does not exist for this recent subject of research, which, nevertheless, already counts with a considerable number of published papers.

Afterwards, we focus on a recently proposed semantics for hybrid integration of rules and ontologies – MKNF well-founded semantics [34] – that has some desirable decidability and complexity properties, providing implementations for such semantics. This is an important contribution as our two implementations represent the first implementations for knowledge bases that combine nonmonotonic rules and DL ontologies, where the combination is such that both the rules can refer to predicates in the ontology, and the ontology can refer to predicates defined in the rules.

As we shall see in the state of the art survey, the definition of this semantics can be based on a fixpoint operator that, being monotonic, readily provides a bottom-up procedure for constructing the result. Building such a bottom-up implementation for the semantics is not a trivial task since the definition itself is parametric on an inference mechanism for the chosen DL. This bottom-up implementation represents the second intermediate goal of this dissertation.

This work then continues to address its main goal: a prototypical goal-driven implementation for the MKNF well-founded semantics of hybrid knowledge bases.
The already mentioned bottom-up implementation computes, step by step, the whole model of the hybrid knowledge base, i.e. it provides the whole set of propositional atoms that are true, and that are false. In practical cases, specially for the Semantic Web, this is not what is wanted. In fact, it would make little sense to compute the whole model of anything that is related to

the *World Wide* Web. Instead, what one would like to do is to query the knowledge base for a given predicate (or propositional atom) and find out whether that is true or false in it (or possibly, come up with substitutions of variables that make it true). Such implementation relies on top-down/goal-driven procedures. In this thesis we propose and implement such a procedure for hybrid knowledge base under the MKNF well-founded semantics.

These implementations, for feasibility given the available time, do not deal with the whole generality of MKNF well-founded semantics. As mentioned above, the proposal is parametric on any given DL, and a corresponding inference mechanism. Hence, in this thesis we focus on a particular DL: CDF [57], which is an ontology management system complete for $\mathcal{ALCQ}$ DL extended with relational hierarchies and product classes. The choice of CDF is not only related with its relative simplicity and the existence of implementations. It is also due to the fact that it is implemented over XSB [26], which is the rule engine embraced for the development of our solutions. In fact, XSB represents the natural choice for these implementations, as it contains several features not usually found in logic programming systems, particularly the ability to evaluate normal logic programs according to the Well-Founded Semantics.

In summary, the contributions of the thesis are:

- A comprehensive state of the art on the recent subject of combining rules and ontologies.
- A first bottom-up implementation of the MKNF well-founded semantics of hybrid knowledge bases, by fixing a simple DL logic (CDF).
- A goal-driven implementation for hybrid knowledge bases under the MKNF well-founded semantics using SLG resolution interleaved with CDF.

1.3 Structure of this document

The next chapters of this report are organised as follows: in Chapter 2 we introduce some essential preliminaries. First, we recall the domain of logic programming in general, referring to stable models and well-founded models in particular. After we review the ideas behind the Semantic Web and particularly OWL and its underlying theory of Description Logics. The current state of art is presented in Chapter 3 and Chapter 4, which we divide into Monotonic Approaches (Chapter 3) and Nonmonotonic Approaches (Chapter 4) to combining rules and ontologies. Afterwards we move forward to the main goals of this dissertation by presenting two different implementations under the MKNF well-founded semantics (Chapter 5). Finally, in Chapter 6 we conclude this document with some final considerations and the remaining challenges for this work.

2 . Preliminaries

2.1 Logic Programming

Logic Programming (LP) is a family of nonmonotonic Knowledge Representation (KR) formalisms, generally written using some special and restricted syntax, known as rules. Contrary to the first-order logic, LP uses negation as failure, i.e., if we fail to derive an atom p, then it is assumed that $\mathbf{not}\ p$ is true.

An alphabet $\mathcal{A}$ is defined over a language $\mathcal{L}$ as a (finite or countable infinite) disjoint set of constants predicate symbols, and function symbols. A term is defined recursively as either a variable, a constant or a expression of the form $f(t_1, \ldots, t_n)$, where f is a function symbol of $\mathcal{A}$, and the t_is are terms. An atom is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of $\mathcal{A}$. A literal is either an atom or its negation.

A term (resp. atom, literal) is called ground if it does not contain variables. The set of all ground terms of $\mathcal{A}$ is called the Herbrand universe of $\mathcal{A}$. Respectively, the set of all ground atoms of $\mathcal{A}$ is denoted as the Herbrand base $\mathcal{H}$.

A normal logic program is a finite set of rules of the following form:

$$H \leftarrow L_1, \ldots, L_n \qquad (n \geq 0)$$

where H is an atom that represents the head of the rule, $L_1, \ldots, L_n$ are literals, and the comma operator is understood as conjunction.

Interpretations of a given program P can be viewed as "potential worlds" representing possible states of the knowledge. In this document, we consider two different kinds of interpretations: 2-valued and 3-valued interpretations. In the former, each atom can only be either true or false. However, there are many cases in which the knowledge of the world is incomplete, and one needs the ability to describe interpretations in which some atoms are neither true nor false. For these cases a 3-valued interpretation is defined, where an atom can also be undefined.

Definition 2.1. (2-valued interpretation). A 2-valued interpretation I of a normal logic program P is any subset of the Herbrand base $\mathcal{H}$ of P.

Definition 2.2. (3-valued interpretation). A 3-valued interpretation I of a program P is represented by a set:

$$T \cup \mathbf{not}\ F$$

where T and F are disjoint subsets of the Herbrand base $\mathcal{H}$ of P. The set T contains all ground atoms true in I, the set F contains all ground atoms false in I, and the truth value of the remaining atoms is undefined.

Any interpretation can be equivalently viewed as a function $I : \mathcal{H} \rightarrow \mathcal{V}$, where $\mathcal{V} = \{0, \frac{1}{2}, 1\}$ defined by:

- $I(A) = 0$ if $A \in F$
- $I(A) = \frac{1}{2}$ if $A \in U$
- $I(A) = 1$ if $A \in T$

Definition 2.3. A partial interpretation is a consistent set of literals whose atoms are in $\mathcal{H}$. A total interpretation is a partial interpretation that contains every atom of $\mathcal{H}$, or its negation.

A logic programming semantics S is a function assigning, to each logic program P, the set of literals "to infer". For this purpose, several semantics for LP have been considered in practice, where *stable model semantics* and *well-founded semantics* are considered the most accepted ones.

2.1.1 Stable Model Semantics

The definition of *stable model semantics* was presented in [18] and refers to 2-valued logic, where an atom can only have the values true or false. Its basic ideas came from the field of nonmonotonic reasoning formalism. There, literals of the form $\mathbf{not}\ A$ are viewed as default literals that may or may not be assumed.

Formally, stable model semantics are defined as:

Definition 2.4. (Gelfond-Lifschitz operator). Let P be a normal logic program and I a 2-valued interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:

- remove from P all rules which contain a default literal $\mathbf{not}\ A$ such that $A \in I$
- remove from the remaining rules all default literals.

Since $\frac{P}{I}$ is a definite program, it has a unique least model J defined as $\Gamma(I) = J$.

It turns out that fixed points of the Gelfond-Lifschitz operator Γ for a program P are always models of P [5]. This result led to the definition of stable model semantics.

Definition 2.5. (Stable model Semantics). A 2-valued interpretation I of a logic program P is a stable model of P iff $\Gamma(I) = I$.

An atom A of P is true under the stable model semantics iff A belongs to all stable models of P.

One of the main advantages of stable model semantics is its close relationship with known nonmonotonic reasoning formalisms as they can be mapped to Reiter's default extensions [45] as well as Moore's autoepistemic expansions [41].

However, stable model semantics still have some important drawbacks, as presented in [5], that are enumerated next:

- Some programs have no stable models.
- Even for programs with stable models, their semantics do not always lead to the expected intended results. When a program has more than one stable how can we choose the "correct" one?
- The computation of stable models is at-least NP-complete even within simple class of programs, such as propositional logic programs.
- There are situations where a 2-valued interpretation is not expressive enough.

2.1.2 Well-Founded Semantics

The *well-founded semantics* has been first introduced in [17]. It overcomes the problems from stable models semantics by a 3-valued interpretation, and assuring that every program has, a unique partial well-founded (or 3-valued) model that can be computed in polynomial time.

For easing the exposition later in this report, rather than presenting here the original definition of [17] we present an alternative one based on the alternating fixpoint [2].

Definition 2.6. (Γ-operator). Let P be a normal logic program and I an interpretation. The GL-transformation $\frac{P}{I}$ is the program obtained from P by performing the operations:

- remove from P all rules which contain a default literal $\mathbf{not}\ A$ such that $I(A) = 1$;
- replace in the remaining rules of P those default literals $\mathbf{not}\ A$ such that $I(A) = \frac{1}{2}$ by $\mathbf{u}$;
- remove from the remaining rules all default literals.

Definition 2.7. A set of objective literals T generates a partial stable model (PSM) of a normal logic program P iff: (1) $T = \Gamma_P^2(T)$; and (2) $T \subseteq \Gamma(T)$. The partial stable model generated by T is the interpretation $T \cup \mathbf{not}\ (\mathcal{H}(P) - \Gamma(T))$.

Theorem 2.1. ***(Well-Founded Semantics).*** *Every non-contradictory program P has a least partial stable model, the well-founded model of P that can be obtained by an iterative "bottom-up" definition:*

$$
\begin{aligned}
&I_0 = \{\} \\
&I_{\alpha+1} = \Gamma^2(I_\alpha) \\
&I_\sigma = \bigcup\{I_\alpha | \alpha < \sigma\} \qquad \textit{for limit ordinal } \sigma
\end{aligned}
$$

There exists a smallest ordinal λ for the sequence above, such that I_λ is the smallest fixpoint of Γ, and the Well-Founded Model - WFM$(P) = I_\lambda \cup (\mathcal{H}_P - \Gamma(I)_\lambda)$

2.1.3 Relation between Stable Model Semantics and Well-Founded Semantics

The well-founded semantics and the stable model semantics are closely related. It is known that if a logic program P has a 2-valued well-founded model (i.e., a total model) then this is the unique stable model of P [17]. However, it turns out that the reverse is not always true, as there are programs with only one stable model, and nevertheless with a partial well-founded model.

Furthermore, the well-founded semantics is sound w.r.t. the stable models semantics. I.e. if an atom is true (resp. false) in the well-founded model, then it is also true (resp. false) in all stable models.

Consider the following example from [5]:

Example 2.1. *The program P:*

$$a \leftarrow \mathbf{not}\ b$$
$$b \leftarrow \mathbf{not}\ a$$
$$c \leftarrow \mathbf{not}\ d$$
$$d \leftarrow \mathbf{not}\ e$$
$$p \leftarrow a$$
$$p \leftarrow b$$

has two stable models: $I_1 = \{p, a, d\}$ and $I_2 = \{p, b, d\}$, and so, under the stable model semantics, both p and d are true.

On the other hand, the well-founded model of P is $\{d, \mathbf{not}\ e, \mathbf{not}\ c\}$. In it, d is true, e and c are false, and a, b and p are undefined.

2.2 Description Logics

Description logics (DLs) are a family of knowledge representation languages that represent the knowledge of an application domain in a structured and formal way. In fact, DLs provide tools for *describing* relevant concepts and properties of objects and individuals occurring in the domain, as they inherit a formal and logic-based semantic given by a translation into first-order logic.

If on the one hand, the price of using first-order logic as a modelling language is too high, as not only the the structure of the knowledge is destroyed, but also the expressive power is too expensive for obtaining decidable and efficient inference problems. On the other, DLs, as a subset of first-order logic, are empowered with some more expressive operations than propositional logical, but still maintaining efficiency when compared to first-order logic.

The building blocks of DL knowledge bases are *concepts* (or classes), *roles* and *individuals*. Roughly speaking, concepts represent sets of objects, while roles define relationships between

these objects. Finally, individuals express instantiated objects. Concepts such as *Person* are called *atomic* and can be combined with concept constructors to obtain *complex* concepts. For example, the concept $\exists hasFather.Person$ describes those objects that are related through the *hasFather* role with an object from the concept *Person* [9].

Furthermore, DLs are a *family* of logics, and depending on the chosen DL, one is able to construct and express a set of concept constructors. These may be boolean connectivities, existential and universal quantifiers, number restrictions, complex roles, etc. As a result, each DL has different expressivity (and efficiency), and one is responsible to analyse the trade-off correctly in order to choose the more appropriate DL for each case.

A typical DL knowledge base $\mathcal{K}$ comprises two components – a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. Briefly, a TBox contains inclusion axioms or definition of concepts similar to a database scheme, while an ABox is a set of individual assertions of a particular world. These assertions can be decomposed in two kinds: concept assertions of the form $C(a)$ and role assertion with the shape $R(a, b)$.

For their relation with first-order logic, DLs employ open world assumption and, therefore, are monotonic. Under Open World Assumption (OWA), failure on deriving a fact does not imply that the fact is false. For example, assume we only know that Mary is a person. It is not possible to conclude that neither Mary is a student, nor that she is not. We can only conclude that our knowledge of the world is incomplete. In the same way, with monotonicity, adding new information to our knowledge never falsifies a previous conclusion. Hence, including *Student(Mary)* to our knowledge does not change any positive or negative conclusions that we could make previously, but only adds new conclusions.

Today, DLs became the cornerstone of the Semantic Web, providing the logical foundation for the Web Ontology Language – OWL.

2.2.1 $\mathcal{SHOIN}(\mathsf{D})$ and $\mathcal{SHIF}(\mathsf{D})$

Herein we recall the foundations of two Description Logics that are the underpinning of the Web Ontology Language OWL, which is be discussed at the end of this chapter.

Usually, the naming of Description Logics languages correspond to the constructors they provide. In the case of $\mathcal{SHOIN}(\mathsf{D})$:

- $\mathcal{S}$ Role transitivity.
- $\mathcal{H}$ Role hierarchy.
- $\mathcal{O}$ Nominals ("oneOf" constructor).
- $\mathcal{I}$ Role inverses.
- $\mathcal{N}$ Unqualified number restrictions.
- D Datatypes.

The logic $\mathcal{SHIF}(\mathsf{D})$ is slightly less expressive:

- $\mathcal{S}$ Role transitivity.
- $\mathcal{H}$ Role hierarchy.
- $\mathcal{I}$ Role inverses.
- $\mathcal{F}$ Functionality
- D Datatypes.

Here, functionality stands for the specific number restriction $\leq 1R$, which is subsumed by the unqualified number restrictions of $\mathcal{SHOIN}(\mathsf{D})$.

2.2.1.1 Syntax

Let $\mathbf{A}, \mathbf{R}_A, \mathbf{R}_\mathsf{D}$, and $\mathbf{I}$ be pairwise disjoint sets of *concept names, abstract role names, datatypes role names* and *individual names*. The set of $\mathcal{SHOIN}(\mathsf{D})$ concepts is the smallest set that can be built using the constructors in Table 2.1 [32].

The semantics of $\mathcal{SHOIN}(\mathsf{D})$ is given by means of an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consisting of a non-empty domain $\Delta^\mathcal{I}$, disjoint from the datatype domain $\Delta^\mathcal{I}_\mathsf{D}$, and a mapping $\cdot^\mathcal{I}$, which interprets atomic and complex concepts, roles, and nominals according to Table 2.1, where $\sharp$ is set cardinality.

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *satisfies* a $\mathcal{SHOIN}(\mathsf{D})$ axiom under the conditions given in Table 2.1. An interpretation satisfies a knowledge base $\mathcal{K}$ iff it satisfies each axiom in $\mathcal{K}$; $\mathcal{K}$ is *satisfiable* iff there exists such an interpretation, and *unsatisfiable* otherwise. A $\mathcal{SHOIN}(\mathsf{D})$ concept C is satisfiable w.r.t. a knowledge base $\mathcal{K}$ iff there is an interpretation $\mathcal{I}$ with $C^\mathcal{I} \neq \emptyset$ that satisfies $\mathcal{K}$. A concept C is *subsumed* by a concept D w.r.t. $\mathcal{K}$ iff $C^\mathcal{I} \sqsubseteq D^\mathcal{I}$ for each interpretation $\mathcal{I}$ satisfying $\mathcal{K}$. Two concepts are said to be equivalent w.r.t. $\mathcal{K}$. A knowledge base $\mathcal{K}_1$ entails a knowledge base $\mathcal{K}_2$ iff every interpretation of $\mathcal{K}_1$ is also an interpretation of $\mathcal{K}_2$.

The description logic $\mathcal{SHIF}(\mathsf{D})$ is just $\mathcal{SHOIN}(\mathsf{D})$ without the $oneOf$ constructor and with the $atleast$ and $atmost$ constructors limited to 0 and 1.

2.2.2 $\mathcal{ALCQ}$

The description logic $\mathcal{ALCQ}$ is obtained from the well-known Attribute Language with Complements ($\mathcal{ALC}$) [53] by including *qualified number restrictions*. $\mathcal{ALC}$ represents the smallest propositional DL, and many description logics are defined as extensions of $\mathcal{ALC}$ by concepts and role constructs. For example, the $\mathcal{S}$ in $\mathcal{SHOIN}$ and $\mathcal{SHIF}$ is translated to $\mathcal{ALC}$ together with transitivity roles. Consequently, the $\mathcal{ALCQ}$ DL is considerably simpler compared to $\mathcal{SHOIN}$ and $\mathcal{SHIF}$, enjoying from `EXPTime`-complete complexity. This DL is the one employed by CDF, which is the ontology framework used for our implementations, and therefore next we present a brief recall of its basic syntax from [16].

Constructor Name	**Syntax**	**Semantics**
atomic concept $\mathbf{A}$	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
datatypes $\mathbf{D}$	D	$D^{\mathsf{D}} \subseteq \Delta^{\mathcal{I}}_{\mathsf{D}}$
abstract role $\mathbf{R}_A$	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
datatypes role $\mathbf{R}_{\mathsf{D}}$	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{\mathsf{D}}$
individuals $\mathbf{I}$	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
data values	v	$v^{\mathcal{I}} = v^{\mathsf{D}}$
inverse role	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
conjunction	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
negation	$\neg C_1$	$(\neg C_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$
oneOf	$\{o_1, \dots\}$	$\{o_1, \dots\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \dots\}$
exists restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta \mid \forall y : (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
atleast restriction	$\geq nR$	$(\geq nR)^{\mathcal{I}} = \{x \in \Delta \mid \sharp(\{y : (x,y) \in R^{\mathcal{I}}\}) \geq n\}$
atmost restriction	$\leq nR$	$(\leq nR)^{\mathcal{I}} = \{x \in \Delta \mid \sharp(\{y : (x,y) \in R^{\mathcal{I}}\}) \leq n\}$
datatype exists	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \in \Delta \mid \exists y : (x,y) \in U^{\mathcal{I}} \wedge y \in D^{\mathsf{D}}$
datatype value	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \in \Delta \mid \forall y : (x,y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathsf{D}}$
datatype atleast	$\geq nU$	$(\geq nU)^{\mathcal{I}} = \{x \in \Delta \mid \sharp(\{y : (x,y) \in U^{\mathcal{I}}\}) \geq n$
datatype atmost	$\leq nU$	$(\leq nU)^{\mathcal{I}} = \{x \in \Delta \mid \sharp(\{y : (x,y) \in U^{\mathcal{I}}\}) \leq n\}$
datatype oneOf	$\{v_1, \dots\}$	$\{v_1, \dots\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \dots\}$
Axiom Name	**Syntax**	**Semantics**
concept inclusion	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
object role inclusion	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
object role transitiviy	$\mathsf{Trans}(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
datatype role inclusion	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
individual inclusion	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
individual equality	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

Table 2.1 Syntax and semantics of $\mathcal{SHOIN}(\mathsf{D})$

2.2.2.1 $\mathcal{ALCQ}$ Syntax

Let N_C, N_R be non-empty and pair-wise disjoint sets of concept names and role names respectively. The set of $\mathcal{ALCQ}$ concepts is the smallest set such that: (i) every concept name $A \in N_C$ is a concept, and (ii) if C, D are concepts and R is a role in N_R, then $\neg C$, $(C \sqcup D)$, $(C \sqcap D)$, $(\exists R.C)$, $(\forall R.C)$, $(\geq nR.C)$, $(\leq nR.C)$, with n a non-negative integer are also concepts.

An interpretation $I = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of $\Delta^{\mathcal{I}}$, a non-empty set of individuals, called the domain of the interpretation and $\cdot^{\mathcal{I}}$ an interpretation function which maps each atomic concept $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, and each atomic role $R, S \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all $\mathcal{ALCQ}$ concepts the following must hold:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \text{ for all } t \in \Delta^{\mathcal{I}} \text{ such that } \langle s,t \rangle \in R^{\mathcal{I}} \text{ then } t \in C^{\mathcal{I}}\}, \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \text{ there exists } t \text{ such that } \langle s,t \rangle \in R^{\mathcal{I}} \text{ and } t \in C\mathcal{I}\}, \\
(\geq nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \sharp(\ \{y : (x,y) \in R^{\mathcal{I}}\}\ ) \geq n\}, \\
(\leq nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \sharp(\ \{y : (x,y) \in R^{\mathcal{I}}\}\ ) \leq n\}
\end{aligned}
$$

where $\sharp S$ denotes the cardinality of the set S. For an interpretation $\mathcal{I}$, a concept C is satisfiable iff $C^{\mathcal{I}} \neq 0$, meaning that there exists an individual $x \in C^{\mathcal{I}}$ as an instance of C, otherwise, C is unsatisfiable. A concept C_1 is *subsumed* by a concept C_2, written as $C_1 \sqsubseteq C_2$, iff for every interpretation $\mathcal{I}$, $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.

2.3 Semantic Web and OWL

Nowadays our lives depend on a relatively simple artifact - the World Wide Web. The Web content consists mainly of distributed hypertext and is accessed by combining searching keywords and link navigations. This simplicity is in fact the most crucial factor for its popularity, as all kinds of users are able to use it and to participate with their own content.

Now, the explosion and growth of both the range and quantity of Web content has motivated a more efficient way to locate and search resources. If humans have difficulty accessing web contents, this problem is even more serious for automated processes. The Semantic Web aims at machine-understandable web resources, whose information can then be shared and processed by automated tools [29].

The notion of Ontology traces back to the work of Plato and Aristotle, where it represented the study of being or existence and its basic categories and thus studied the structure of the

world.

In the context of the Semantic Web, ontologies are expected to play an important role in helping these automated processes to access information. Particularly, ontologies are metadata schemas, providing structured and controlled vocabulary of concepts and relations. By defining shared and common domain theories, ontologies empower intelligent agents with tools to interpret their meaning in a flexible but yet unequivocally way.

The Web Ontology Language (OWL) appears as a family of languages for modelling ontologies in the Semantic Web. OWL is endorsed by the World Wide Web Consortium (W3C) and can be divided in three different semantics: OWL-Lite, OWL-DL and OWL-Full.

The first two variants are based on DLs and correspond syntactically to $\mathcal{SHIF}$(D) and $\mathcal{SHOIN}$(D), respectively, making OWL-Lite a subset of OWL-DL. OWL-Full was designed and modelled to provide full compatibility with RDF Schema which makes it undecidable and consequently, unattractive to implement [42].

OWL-DL it is the mostly used sublanguage as it offers a high level of expressivity preserving decidability. As mentioned before, $\mathcal{SHOIN}$(D) provides full negation, disjunctive, and (in a restricted form) universal and existential quantification of variables.

The characteristics of OWL have triggered the development of tools and reasoning systems that contributed to its use, not only in the Semantic Web, but as a popular language for ontology development in fields like Biology, Medicine, Geography, Geology, Astronomy, Agriculture or Defense. In fact, the style of modelling supported by OWL has proven to be particularly suitable for modelling taxonomic knowledge as it is possible to use open-world semantics to express truths about known and unknown individuals, providing an elegant way of modelling incomplete information.

Next, we focus on two different extensions made for OWL, which emerged from the need to overcome problematic restrictions identified by its users.

2.4 OWL Extensions

2.4.1 OWL 1.1

The conservative initial design of the OWL made some features and reasoning methods to be excluded. Some of these, like *qualified number restrictions*, were already supported by DL systems when OWL was designed. Others, such as *complex role inclusion axioms*, could now be supported (at least in part) as a result of recent advances in DL theory.

This idea led to the development of an incremental extension of OWL, provisionally called OWL 1.1, which would exploit these recent developments in order to provide a more expressive language, but still, retaining the OWL's desirable computational properties [22]. It incorporates the entire OWL DL syntax, only providing extensions to it. Hence, any legal OWL DL ontology remains a legal OWL 1.1 ontology.

The features added in OWL 1.1 fall into four main categories, where the change of the DL

employed is considered the most important.

1. **Syntactic Sugar.** OWL 1.1 provides two constructs that are simply syntactic sugar, to make some common idioms easier to write: `DisjointUnion` and negative property membership assertions, `NegativeObjectPropertyAssertion` and `NegativeDataPropertyAssertion`.

2. **SROIQ.** OWL 1.1 provides extra DL expressive power, upgrading from $\mathcal{SHOIN}$ DL that underlies OWL DL to $\mathcal{SROIQ}$ DL. Particularly, $\mathcal{SROIQ}$ consider complex roles inclusion axioms of the form $R \circ S \sqsubseteq R$ or $S \circ R \sqsubseteq R$ to express propagation of one property along another one. Furthermore, it is extended with reflexive, symmetric, transitive, and irreflexive roles; as well as with disjoint roles, universal role, and constructs $\exists R.Self$, allowing, for instance, the definitions of concepts such as a "narcist". Finally, it considers also negated role assertions in ABoxes and qualified number restrictions [30].

   Besides a TBox and an ABox, $\mathcal{SROIQ}$ provides a so-called *RBox* to gather all statements concerning roles. It is designed to be of similar practicability as $\mathcal{SHIQ}$, supported by many reasoners like KAON2[1] or RacerPro [2]. As a result, the tableau algorithm for $\mathcal{SROIQ}$ is roughly a combination of the algorithms for $\mathcal{RIQ}$ and $\mathcal{SHOIQ}$.

3. **Datatypes.** OWL 1.1 allows user-defined datatypes, using a mechanism similar to what is used in Protégé[3].

4. **Metamodelling constructors.** OWL 1.1 provides Metamodelling based on the Meta Object Facility (MOF) [4]. MOF uses the Unified Modelling Language (UML) as a visual notation. Particularly, MOF uses a very simple form of UML class diagrams that are expected to be intuitively understandable to readers that are familiar with the basic concepts of object-oriented systems, even if they are not familiar with UML.

From the start, OWL 1.1 was intended to be an easy and incremental improvement to OWL. However, it lead a movement towards a larger and deeper extension called OWL 2.0 that is explained next.

2.4.2 OWL 2.0

OWL 2.0 is being developed by the W3C Web Ontology Working Group and it is expected to be ready during 2009. In order to address the problems of the previous version, OWL 2 extends the OWL 1 with a small but useful set of features that have been requested by users. The new features include extra syntactic sugar, additional property and qualified cardinality constructors, extended datatypes support, simple metamodelling, and extended annotations. OWL 2 is

[1] `http://kaon2.semanticweb.org/`
[2] `http://www.racer-systems.com/`
[3] `http://protege.stanford.edu/`
[4] `http://www.omg.org/docs/formal/06-01-01.pdf`

intended to be a superset of OWL 1, which means that OWL 2 will be backward compatible with OWL 1.

The OWL 2 specification is divided into several profiles – OWL 2 EL, OWL 2 QL and OWL 2 RL – each of which provides different expressive power and targets different application scenarios [21]. The choice of profile will depend on the structure of the ontologies used in the application as well as the reasoning tasks to be performed.

Apart from the profiles specified here, one can also define another possible profile of OWL 2. For example, it is possible to define a subset of the OWL 2 that corresponds to OWL 1 Lite or OWL 1 DL.

2.4.2.1 OWL 2 EL

OWL 2 EL is based on the $\mathcal{EL}^{++}$ family of description logics, which have been designed to allow for efficient reasoning with large terminologies [8]. It can be seen as a syntactic fragment of OWL 1.1 DL that admits sound and complete reasoning in polynomial time [20].

The main advantage of $\mathcal{EL}^{++}$ is that it combines sufficient tractability with expressive power to be employed in many important applications of ontologies. For instance, the Thesaurus of the National Cancer Institute (NCI)[5] ontology which comprises about 25000 classes, or the Gene Ontology[6] which formalises terms relating to genes and gene productions, fall within $\mathcal{EL}^{++}$. In the same way, more than 95 % of the axioms of the GALEN[7] ontology can also be expressed in $\mathcal{EL}^{++}$. $\mathcal{EL}^{++}$ is discussed in more detail on Section 3.4.2.

2.4.2.2 OWL 2 QL

OWL 2 QL is based on the DL-Lite$_R$ family of DLs and has been designed to allow efficient reasoning with large amounts of data structured with a relatively simple schemata. DL-Lite$_R$ is an expressive DL containing the intersection of RDFS and OWL 2. It does not require the UNA, since this assumption has no impact on the semantic consequences of a DL-Lite$_R$ ontology. Thus, this choice avoids practicable problems involved in the explicit axiomatisation of UNA.

In this profile, query answering is the most important reasoning task and can be performed in `LogSpace` with respect to the size of the asserted data. Like in OWL 2 EL, there are polynomial time algorithms for consistency, subsumption and classification reasoning.

The expressive power of this profile is necessarily quite limited, although it includes most of the main features of conceptual model, such as UML class diagrams and ER diagrams.

2.4.2.3 OWL 2 RL

OWL 2 RL is aimed for applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade

[5] `http://www.nci.nih.gov/cancerinfo/terminologyresources`
[6] `http://www.geneontology.org/`
[7] `http://www.openclinical.org/prj_galen.html`

the full expressivity of the language for efficiency, and RDF(S) applications that need some added expressivity.

OWL 2 RL allows for most constructors of OWL. However, in order to allow rule-based implementations of reasoning, the way these constructors can be used in axioms has been restricted. These restrictions ensure that a reasoning engine only needs to reason with the individuals that occur explicitly in the ontology. This principle has been used before in DL-safe rules.

Furthermore, OWL 2 RL specification also provides a set of first-order implications that can directly be applied to an RDF graph in order to derive the relevant consequences. These implications are reminiscent of the pD* semantics for OWL 1 and provide a useful starting point for the implementation of forward-chaining reasoners for OWL 2 RL.

This way, it is possible to say that its design was inspired by Description Logic Programs[23] and pD*[60].

3 . Monotonic Approaches

Monotonicity indicates that learning a new piece of knowledge cannot reduce the set of what is known, meaning that adding a formula to a theory never produces a reduction of its set of consequences.

Monotonic approaches are relatively simple approaches that, like OWL, only support monotonic inference. Thus, they achieve a higher expressivity by allowing some special types of formulas that are common in rules, connected with constructs of the DLs. Although, not dealing with closed world reasoning, we shall see that this combination cannot be done straightforwardly at the price of compromising decidability. As such, since in the end all can be translated into FOL, these approaches are not aiming at presenting an intuitive combined semantics, but rather achieving expressivity while still avoiding raising (too much) the computational complexity.

Next, to illustrate the power of integrating rules and ontologies in a monotonic solution, we present Example 3.1. Furthermore, since each approach has its particular expressivity, we will use this example along this section to point out the differences between each solution as well as show which axioms and rules can be expressed in each case.

Example 3.1. *Consider a simple example concerning a Thai Restaurant. In this case, we could use Description Logics to model characteristics of Thai dishes, for instance, stating that Thai Curry is a Thai Dish and that every Thai Curry contains peanut oil. Moreover, we could add an ABox expressing that Sebastian ordered a Thai Curry dish.*

$$thaiCurry \sqsubseteq ThaiDish \tag{3.1}$$
$$ThaiDish \sqsubseteq Dish \tag{3.2}$$
$$thaiCurry \sqsubseteq \exists contains.\{peanutOil\} \tag{3.3}$$
$$T \sqsubseteq \forall orderedDish.Dish \tag{3.4}$$
$$sebastian : \exists orderedDish.thaiCurry \tag{3.5}$$

In addition, we could include some rules to our knowledge base with some statements that can't be expressed in a Description Logic Language as 3.6 *and* 3.7*. The first rule states that people that are allergic to nuts dislike nut products, whilst the latter expresses that people which order a dish they dislike are unhappy. From this example, and by adding facts* 3.8 *and* 3.9*, we can conclude that Sebastian is unhappy.*

$$dislikes(x, y) \leftarrow nutAllergic(x), nutProduct(y). \tag{3.6}$$
$$unhappy(x) \leftarrow orderedDish(x, y), dislikes(x, y). \tag{3.7}$$
$$nutAllergic(sebastian). \tag{3.8}$$
$$nutProduct(peanutOil). \tag{3.9}$$

3.1 $\mathcal{AL}$-log

$\mathcal{AL}$-log, Attributive Language and Datalog, [13] represents the first formal hybrid reasoning system. This proposal is monotonic and integrates knowledge bases expressed in one of the simplest kinds of DLs – $\mathcal{ALC}$ and positive Datalog programs. Afterwards, it was extended based on the use of Disjunctive Datalog by [48].

$\mathcal{AL}$-log can be divided into two subsystems called structural and relational. The former allows the definition of structural knowledge about concepts, roles and individuals, while the latter provides a suitable extension of Datalog in order to express relational knowledge about objects described in the structural component. The interaction between the two components is obtained by allowing constraints within Datalog clauses, thus requiring the variables in the clauses to range over the set of instances of a specified concept.

3.1.1 The Structural Subsystem

The structural subsystem contains the definition of what is called an $\mathcal{ALC}$ knowledge base. This knowledge base is itself structured in a two-component system, the *intensional* component and the *extensional* component. The intensional component, $\mathcal{T}$, consists of concept hierarchies and *is-a* relations between them, namely, *inclusion statements* of the form $C \sqsubseteq D$, where C and D are two arbitrary concepts. Moreover, the extensional component, $\mathcal{A}$, specifies instance-of relations, e.g. *concept assertions* of the form $a : C$ and *role assertions* of the form aRb.

The features of the structural subsystem of $\mathcal{AL}$-log can be characterized as follows. The $\mathcal{ALC}$ knowledge base Σ, is a pair $\Sigma = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ (the intensional level) is a set of inclusions and $\mathcal{A}$ (the extensional level) is a set of assertions.

Citing from [13], an interpretation $\mathcal{I}$ is a model of Σ if it is both a model of $\mathcal{T}$ and a model of $\mathcal{A}$. By virtue of the unique name assumption, an $\mathcal{O}$-interpretation for Σ is an interpretation such that $\mathcal{O} \subseteq \Delta^{\mathcal{I}}$ and for each $a \in \mathcal{O}$, $a^{\mathcal{I}} = a$. An $\mathcal{O}$-model is an $\mathcal{O}$-interpretation that is a model. Σ is *satisfiable* if it has a model.

3.1.2 The Relational System

The relational part of $\mathcal{AL}$-log allows one to define Datalog programs enriched with constraints of the form $s : C$ where s is either a constant or a variable, and C is an $\mathcal{ALC}$-concept. The symbol & separates constraints from Datalog atoms in a clause.

Definition 3.1. A constrained Datalog clause is an implication of the form:

$$\alpha_0 \rightarrow \alpha_1, \ldots, \alpha_m \& \gamma_1, ..., \gamma_n$$

where $m \geq 0$, $n \geq 0$, α_i are Datalog atoms and γ_j are constraints. A constrained Datalog program Π is a set of constrained Datalog clauses.

For example, the following constrained Datalog clause [13]:

`curr(X,Z) :- exam(X,Y),subject(Y,Z)&X:Student,Y:Course,Z:Topic`

states that a student X has a topic Z in its curriculum if there exists a course Y such that X has passed the exam and Z is a topic of Y.

3.1.3 Semantics of $\mathcal{AL}$-log Knowledge Bases

An $\mathcal{AL}$-log knowledge base $\mathcal{B}$ is the pair (Σ, Π) where Σ is an $\mathcal{ALC}$ knowledge base and Π is a constrained Datalog program. In order to be acceptable, the knowledge base must satisfy the following conditions:

- The set of Datalog predicate symbols appearing in Π is disjoint from the set of concept and role symbols appearing in Σ.
- The alphabet of constants in Π coincides with the alphabet $\mathcal{O}$ of the individuals of Σ. Furthermore, every constant in Π also appears in Σ
- For each clause in Π, each variable occurring in the constraint part occurs also in the Datalog part.

Definition 3.2. An interpretation J for $\mathcal{B} = (\Sigma, \Pi)$ is defined as the union of an interpretation I for Σ and a Herbrand interpretation H for Π_D, where Π_D is obtained from Π by deleting the DL predicates.

An interpretation $J = (I, H)$ is a model of $\mathcal{B}$ if I is a model of Σ and for each ground case in Π, either one of the DL predicates is not satisfied by J or the non-DL part of the clause is satisfied by J.

These properties allow for the extension of terminology and are thus related to the notion of *substitution* from Datalog to $\mathcal{AL}$-log in a straightforward manner [39]. Reasoning for $\mathcal{AL}$-log knowledge bases relies on *constrained SLD-resolution*, which is an extension of SLD-resolution with tableau calculus for $\mathcal{ALC}$ to deal with constraints. A constrained SLD-derivation for a query Q_0 in $\mathcal{B}$ is a derivation constituted by:

- a sequence of queries $Q_0, ..., Q_n$;
- a sequence of constrained Datalog clauses $E_1, ..., E_n$;
- a sequence of substitutions $\theta_1, ..., \theta_n$

Where $i \in 0, ...n-1$, Q_{i+1} is the resolvent of Q_i and E_{i+1} with substitution θ_{i+1}. n is called the length of the derivation.

Definition 3.3. Let $Q^{(O)}$ be a query $\leftarrow \beta_1, ...m, \beta_m \& \gamma_1, ..., \gamma_n$ to an $\mathcal{AL}$-log knowledge base $\mathcal{B}$. A constrained SLD-refutation for $Q^{(O)}$ in $\mathcal{B}$ is a finite set $\{d_1, ..., d_s\}$ of constrained SLD-derivations for $Q^{()}$ in $\mathcal{B}$ such that:

1. for each derivation d_i, $1 \leq i \leq s$, the last query $Q^{(n_i)}$ of d_i is a constrained empty clause;
2. for every model J of $\mathcal{B}$, there exists at least one derivation d_i, $1 \leq i \leq s$, such that $J \models Q^{(n_i)}$

This definition comes from the fact that a derivation of the empty clause with associated constraints does not represent a refutation, because what is actually inferable from such derivation is that the query is true in those models of $\mathcal{B}$ that satisfy the constraints. Therefore, in order to answer to a query it is necessary to collect enough derivations ending with a constrained empty clause, such that every model of $\mathcal{B}$ satisfies the constraints associated with the final query of at least one derivation.

Constrained SLD-resolution is decidable and runs in single non-deterministic exponential time [39]. An answer σ to a query Q is a *computed answer* if there exists a constrained SLD-refutation for $Q\sigma$ in $\mathcal{B}$ ($\mathcal{B} \vdash Q\sigma$). The set of computed answers is called the *success set* of Q in $\mathcal{B}$. Furthermore, given any query Q, the success set of Q in $\mathcal{B}$ coincides with the stable model of Q in $\mathcal{B}$, which provides an operational means for computing correct answers to queries.

Recall Example 3.1 presented earlier. This example cannot be fully stated in the particular case of $\mathcal{AL}$-log, since this solution only allows axioms that are expressible in $\mathcal{ALC}$ DL. This is in fact, a considerable expressibility restriction as $\mathcal{ALC}$ represents one of the simplest DL. Consequently, in Example 3.1 it is not possible to express Axiom 4.3: $thaiCurry \sqsubseteq \exists contains.\{peanutOil\}$ as $\mathcal{ALC}$ does not support nominals.

3.2 Semantic Web Rule Language

Semantic Web Rule Language (SWRL) is a proposal, supported by W3C, for extending the syntax of OWL-Lite and OWL-DL with Unary/Binary Datalog RuleML [31]. SWRL is a simply extension of OWL DL model theory, with no nonmonotonic features involved. It extends the set of OWL axioms by including Horn-like rules as well as empowers the same rules to be combined with OWL knowledge bases.

As most other rule languages, SWRL rules are written as antecedent-consequent pairs. The intended meaning can be read as: whenever the conditions specified in the antecedent (body) hold, then the conditions specified in the consequent must also hold.

Both the antecedent and consequent consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e., satisfied by every interpretation), so the consequent must also be satisfied by every interpretation. Symmetrically, an empty consequent is treated as trivially false (i.e., not satisfied by any interpretation). Multiple atoms are treated in separate as a conjunction, which means that all atoms must be satisfied. A rule can be identified by a URI reference.

Atoms can be shaped as $C(x)$, $P(x)$, $sameAs(x, y)$, $differentFrom(x, y)$ or $builtIn(r, x, ...)$, where C is an OWL description or data range; P is an OWL property;r is a built-in relation; x and y are either variables, OWL individuals or OWL data values. In the context of OWL Lite, descriptions of the form $C(x)$ may be restricted to class names.

Roughly speaking, an atom $C(x)$ holds if x is an instance of the class description of data range C; an atom $P(x, y)$ holds if x is related to y by property P; an atom $sameAs(x, y)$ holds if x is interpreted as the same object as y; an atom $differentFrom(x, y)$ holds if x and y are interpreted as different objects; finally, $builtIn(r, x, ...)$ holds if the built-in relation r holds on the interpretation of the arguments.

The semantics for SWRL is a straightforward extension of the semantics for OWL DL. The idea is to define bindings, which are extensions of OWL interpretations that also map variables to elements of the domain. A rule is satisfied by an interpretation only if every binding that satisfies the antecedent also satisfies the consequent.

To illustrate, consider the following example from [31]. To state that the combination of the `hasParent` and `hasBrother` properties implies the `hasUncle` property, the rule can be written as:

```
hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)
```

In the abstract syntax the rule would be written like:

```
Implies(Antecedent(hasParent(I-variable(x1) I-variable(x2)),
                   hasBrother(I-variable(x2) I-variable(x3)))
        Consequent(hasUncle(I-variable(x1) I-variable(x3))))
```

Contrary to other solutions that propose an intermediate language contained within the intersection of DLs and rules, SWRL presents itself as a comfortable unified platform to express both in its intrinsic style. In fact, Example 3.1 presented earlier can be completely expressed in SWRL without restrictions. However, by allowing the fully union between DLs and rules, SWRL becomes undecidable. This way, SWRL retains the full power of OWL DL but faces the standard trade off for expressivity: loss of efficiency.

SWRL is thus seen as a naive combination of OWL and rules which, nevertheless, was very important to understand the need of concepts such as DL-Safety, that are explained later in Chapter 4.

3.3 RIF

The W3C Rule Interchange Format (RIF) Working Group [24] is developing since December 2005 an effort to define a standard to help the exchange of rule sets among different systems. Contrary to other approaches, RIF itself does not provide neither a translation algorithm nor an explicit mapping between rule languages. The idea behind it relies on the concern that a Web standard for just one rule language could block the progress in this area of research [12].

Moreover, RIF intends to be an *interchange format* for rules in alignment with the existing standards in the Semantic Web architecture stack. However, creating a generally accepted interchange format is not a trivial task. First, there are different understandings of what a rule is (deductive rules, normative rules, production rules, reactive rules, etc.). Second, even with the

same category of rules, systems often use incompatible semantics and syntaxes. As a solution, RIF includes a framework of concepts, represented as tags in a markup language, that are used to provide information about the meaning of a well-formed formula in a rule language. For rule authors who wish to make their rules accessible across languages and platforms, the more completely and precisely they tag their creations using RIF, the more likely their rules will be capable of being automatically translated correctly.

The idea is that RIF defines a standard way to express the structure of each rule language. Then, for each language, one would only have to write two translation algorithms: one from the language to the RIF and vice-versa.

One of the main goals of this approach is to reuse and apply existing technologies and standards, even when it makes the design job harder. As a consequence, RIF aims to obtain the desired standardisation by interoperating with XML, RDF, SPARQL and OWL.

3.3.1 RIF Frameworks

The design of RIF is intended to provide general extensibility, as well as support for both backward and forward compatibility. To support this idea, RIF's architecture is based on the idea of having multiple dialects, each with its own XML syntax and well-defined semantics. Each dialect is a collection of components that work together, forming an interlingua [25].

To ensure a maximum degree of coherence among the various dialects, RIF defines *frameworks*, i.e., general formalisms that can be specialised to particular dialects. Currently, the RIF working group is focused on two families of dialects: *logic based dialects* and *production rule dialects*. The former family covers rule systems that are based on logic programming, deductive database paradigms and first-order logic, while the latter is intended to account for many commercial condition-action rule systems which is not related to the focus of this work and, consequently, is not covered in this report.

3.3.2 RIF-FLD

The RIF Framework for Logic-based Dialects (RIF-FLD) is a formalism for specifying all logic-based dialects of RIF. This framework is very general and captures most of the popular logic rule languages found in Databases, Logic Programming, and on Semantic Web. The design of RIF-FLD envisages the future standard logic dialects to be a specialisation or an extension from FLD. RIF-FLD can be divided in three main components: syntactic framework, semantic framework and XML serialisation framework that are explained next.

3.3.2.1 Syntactic framework

The syntactic framework defines the mechanisms for specifying the formal presentation syntax of RIF's logic dialects. This syntax is not intended to be concrete for all dialects, as it deliberately leaves out details such as the delimiters of the various syntactic components, escape symbols, parenthesising, precedence of operators, etc. For instance, RIF-BLD (Basic Logic

Dialect) is a specialisation of RIF-FLD, and it deliberately omits such details. This framework defines six types of RIF terms that RIF dialects can choose to support:

- *Constants and variables*, common in most logic languages.
- *Positional terms*, which are usually in first-order logic. However they are defined in a slightly more general way in order to enable dialects with higher-order syntax.
- *Terms with named arguments*, that are similar to positional terms, except from the fact that each argument of a term is named and the order of the arguments is irrelevant. Terms with named arguments generalize the notion of rows in relational tables, where column headings correspond to argument names.
- *Frame* term, represents an assertion about an object and its properties. These terms correspond to molecules of F-logic.
- *Classification* terms, used to define the subclass and class membership relationships. Like frames, they are also borrowed from F-logic
- *Equality* terms, used to equate other terms

3.3.2.2 Semantic framework

The semantic framework defines the notion of a *semantic structure* or *interpretation*. Thus, it is used to interpret formulas and to define logical entailment employing a number of mechanisms:

- *Truth values.* RIF-FLD accommodates dialects that support reasoning with inconsistent and uncertain information. This way, it is designed to deal with multi-valued logics, including the values true, false and possibly others.
- *Data types.* A symbol space whose symbols have a fixed interpretation in any semantic structure is called a data type. For instance, symbols in the symbol space `xsd:string` are always interpreted as sequences of unicode characters.
- *Entailment.* This notion is fundamental to logic-based dialects. Given a set of formulas F, entailment determines which other formulas necessarily follow from F. This is the main mechanism underlying query answering in databases, logic programming, and the various reasoning tasks in Description Logic.

3.3.2.3 XML serialisation framework

Finally, XML serialisation framework defines the general principles for mapping the presentation syntax of RIF-FLD to the concrete XML interchange format. It includes a specification of the XML syntax of RIF-FLD as well as the associated XML Schema document; and a specification of a one-to-one mapping from the presentation syntax of RIF-FLD to its XML syntax.

As a result, any valid XML document for a logic-based RIF dialect must also be a valid XML document for RIF-FLD. In terms of the presentation-to-XML syntax mappings, this means that each mapping for a logic-based RIF dialect must be a restriction of the corresponding mapping for RIF-FLD.

3.3.3 RIF-BLD

As stated before, the Basic Logic Dialect of the Rule Interchange Format (RIF-BLD) is a specialisation from the syntax of the RIF-FLD and, currently, is the only which already has concrete semantics. It corresponds to a syntactic variant of Horn rules with equality and with a standard first-order semantics which most available rule systems can process. It has a number of extensions to support features such as objects and frames as in F-logic [33]. RDF-BLD is a *web language* as it supports the use of IRIs as identifiers for concepts, XML Schema data types, as well as a formalised compatibility with RDF and OWL. Nevertheless, RIF is designed to enable interoperability among rule languages in general, and its uses are not limited to the Web.

RIF-BLD supports the following types of formulas:

- *RIF-BLD condition* is an atomic formula, a conjunctive or disjunctive combination of atomic formulas, or an external atomic formula. All these can optionally have existential quantifiers.

- *RIF-BLD rule* is a universally quantified RIF-FLD rule with the following restrictions:

  - The conclusion of the rule is an atomic formula or a conjunction of atomic formulas.
  - None of the atomic formulas mentioned in the rule conclusion are externally defined.
  - The premise of the rule is a RIF-BLD condition.
  - All free (non-quantified) variables in the rule must be quantified with `Forall` outside of the rule

- *Universal fact* is a universally quantified atomic formula with no free variables.

- *RIF-BLD group* is a RIF-FLD group that contains only RIF-BLD rules, universal facts, variable-free implications, variable-free atomic formulas, and RIF-BLD groups.

- *RIF-BLD document* is a RIF-FLD document that consists of directives and a RIF-BLD group formula.

As a monotonic approach, RIF-BLD does not allow any kind of reasoning combining open-world and closed-world assumption, since negation (neither classical nor default) is not supported by RIF-BLD in either the rule conclusion or the premise.

3.4 ELP

3.4.1 Description Logic Programs

Description Logic Programs (DLP) is a Knowledge Representation (KR) that is contained within the intersection of DL and LP [23]. In contrast to SWRL, DLP restricts the syntax of the supported OWL DL fragment to those axioms expressible in Horn rules [14]. This representation is used as an intermediate KR which enables the combination between rules and ontologies. This technique is named DLP-*fusion* and it allows the bidirectional mapping of premises and inferences from DLP fragment of DL to LP, as well as from the DLP fragment of LP to DL.

DLP-*fusion* enables one to "build rules on top of ontologies" as it enables the rule to have access to DL ontological definitions for vocabulary primitives. Reversely, this technique also enables to "build ontologies on top of rules" by providing ontological definitions to be supplemented by rules, or imported into DL from rules. Furthermore, it allows efficient LP inferencing algorithms/implementations to be exploited over large-scale DL ontologies.

DLP provides a significant degree of expressiveness, substantially greater than the RDFS fragment of Description Logics. DLP also capture a substantial fragment of OWL by including the whole OWL fragment of RDFS, simple frame axioms and more expressive property axioms.

As for performance, DLP enjoys polynomial data complexity and `ExpTime` combined complexity. DLP also provides a flexible choice for the future as it is a common fragment of major paradigms, in principle, it is compatible with whatever paradigm will turn out to be more popular. Extensions made for any more general language can be adopted for the fragment in a straightforward manner. Modelling and reasoning tools available for OWL or F-Logic can naturally deal with DLP, and interoperability is guaranteed to the largest extent possible.

However, regarding Example 3.1 presented earlier, since DLP is obtained in the intersection of DLs and Datalog, it is only possible to express axioms and rules that fall within this intersection. As a result, neither of these could be expressed:

$$sebastian : \exists orderedDish.thaiCurry$$
$$dislikes(x, y) \leftarrow nutAllergic(x), nutProduct(y).$$
$$unhappy(x) \leftarrow orderedDish(x, y), dislikes(x, y).$$

In fact, the DLP does not increase the expressive power of OWL, which is one of the main objectives for introducing rules [7].

3.4.2 ELP

ELP appears as a response to the ongoing standardisation of the new Web Ontology Language OWL 2, reconciling $\mathcal{EL}^{++}$ and DLP in a new novel rule based KR.

This approach can thus be seen as an extension of both formalism, which, however, still preserves polynomial time by limiting the interactions between the expressive features. In another way, ELP can also be defined as a decidable subset of SWRL as it is possible to indirectly express such rules by means of the expressive features provide by $\mathcal{SROIQ}$. Furthermore, large parts of ELP can still be regarded as a subset of $\mathcal{SROIQ}$.

The reasoning algorithms applied are based on a polynomial reduction of ELP knowledge bases to a specific kind of Datalog programs that can be evaluated also in polynomial time. Since the Datalog reduction as such is comparatively simple, it is important to understand the implementation strategy for the $\mathcal{EL}^{++}$ profile of OWL 2. Particularly, ELP extends the DL $\mathcal{EL}^{++}$ with local reflexivity, concept products, conjunctions of simple roles and limited range restrictions.

Definition 3.4. An $\mathcal{EL}^{++}$ concept expression is a $\mathcal{SHOQ}$ concept expression that contains only the following concept constructors: $\sqcap, \exists, \bot, \top$, as well as nominal concepts a

Besides the possibility of reusing optimisation methods from deductive databases, the compilation of $\mathcal{EL}^{++}$ to Datalog also provides a practical approach for extending $\mathcal{EL}^{++}$ with DL-Safe rules. Moreover, this integration of DL-Safe rules is not trivial since, in the absence of inverse roles, it cannot be achieved by the usual approach for “rolling-up" nested expressions and termination of the modified transformation is less obvious. However, this integration is crucial. Indeed, even though reasoning with $\mathcal{EL}^{++}$ is possible in polynomial time, extending it with further forms of rules (even if restricting to Datalog) readily leads to undecidability.

In addition to various forms of DL-safe rules, ELP also allows for special rules of the form $R(x, y) \rightarrow C(y)$ expressing *range restrictions* on the role R. Such restrictions are neither DL-safe Datalog nor DL rules, and, in general, they also lead to undecidability of $\mathcal{EL}^{++}$. Nevertheless, it has recently been observed that range restrictions can still be admitted under certain conditions [8].

Definition 3.5. A rule $B \rightarrow H$ is a basic ELP rule if:

- $B \rightarrow H$ is an extended $\mathcal{EL}^{++}$ rule
- the rule $B' \rightarrow H'$ obtained from $B \rightarrow H$ by replacing all safe variables by some individual name is a DL rule.

An ELP rule base RB is a set of basic ELP rules together with range restriction rules of the form $R(x, y) \rightarrow C(y)$, that satisfies the following condition:

- If RB contains rules of the form $R(x, y) \rightarrow C(y)$ and $B \rightarrow H$ with $R(t, z) \in H$ then $C(z) \in B$.

Whenever a set of range restriction rules satisfies the above condition for some set of ELP rules, we say that the range restrictions are admissible for this rule set.

A rule $B \rightarrow H$ is an ELP_n rule for some natural number $n > 2$ if it is either an ELP rule, or a DL-safe Datalog rule with at most n variables.

As mentioned before, ELP subsumes other tractable languages like DLP, a formalism introduced as the intersection of DL $\mathcal{SHOIQ}$ and Datalog. DLP can also be generalised using DL rules. A DLP head concept is any $\mathcal{SHOQ}$ concept expression that includes only concept names, nominals, $\sqcap, \top, \bot$, and expressions of the form $\leq 1R.C$, where C is an $\mathcal{EL}^{++}$ concept expression. A DLP rule $B \rightarrow H$ is an extended DL rule such that all concept expressions in B are $\mathcal{EL}^{++}$ concept expressions, and all concept expressions in H are DLP head concepts.

The combination of DLP and $\mathcal{EL}$ is `ExpTime` complete. Yet, DLP and $\mathcal{EL}^{++}$ inferences can be recovered in ELP without losing tractability. As said earlier, ELP can be seen as extension both of DLP and $\mathcal{EL}^{++}$, which follows the next theorem [8]:

Theorem 3.1. *Consider any ground atom α of the form $C(a)$ or $R(a, b)$. Given a DLP rule base RB and an $\mathcal{EL}^{++}$ description logic knowledge base KB, one can compute an ELP rule base RB' in linear time, such that: If $RB \models \alpha$ then $RB \cup KB \models \alpha$.*

However, the resulting ELP rule base entails all individual consequences of RB and KB, both not all consequences of their (unsafe) union. Consequently, it provides a means for combining $\mathcal{EL}^{++}$ and DLP in a way that prevents intractability, while still allowing for a controlled interaction between both languages. Simple atomic concept and role inclusions, for instance, can always be considered as $\mathcal{EL}^{++}$ axioms, and all concept subsumptions entailed from the $\mathcal{EL}^{++}$ part of a combined knowledge base do also affect classification of instances in the DLP part. This way, DLP gains the terminological expressivity of $\mathcal{EL}^{++}$ while still having available specific constructs that may only affect the instance level.

To illustrate, remember Example 3.1 presented above. This particular example can be fully expressed in ELP. However, in order to maintain tractability, it requests for a safe interaction only in Rule 3.7: $unhappy(x) \leftarrow orderedDish(x, y), dislikes(x, y)$.

3.4.3 Polytime ELP Reasoning with Datalog

Reasoning in ELP is supported by a transformation of the rule base into a Datalog program. This transformation is only be briefly discussed next, and we refer [36] for more details. ELP considers three disjoint sets of *individual names* N_I, *concept names* N_C, and *role names* N_R. N_R is assumed to be the union of two disjoint sets of *simple roles* N_R^S and *non-simple roles* $N_R^{\cap}$.

- The first step consists in transforming $\mathcal{EL}^{++}$ rule base RB into an equisatisfiable $\mathcal{EL}^{++}$ RB' in normal form. An $\mathcal{EL}^{++}$ rule base RB' is in normal form if all concept atoms in rule bodies are either concept names, $\top$, or nominal concepts; all variables in a rule's head also occur in its body, and all rule heads contain only atoms of the following forms:

$$A(t) \qquad \exists R.B(t) \qquad R(t, u)$$

  where $A \in N_C \cup \{\{a\} | a \in N_I\} \cup \{\bot\}, B \in N_C, R \in N_R$, and $t, u \in N_I \cup V$. This transformation is achieved in polynomial time.

- It is now possible to define a Datalog program $\overline{P}$(RB), which is obtained as follows:

  1. For each individual a occurring in RB the program $\overline{P}$(RB) contains rules $\rightarrow C_a(a)$ and $C_a(x) \rightarrow R_{\approx}(x, a)$.
  2. For each concept name C and role name R occurring in $\overline{P}$(RB), the program $\overline{P}$(RB) contains the rules:

$$\begin{aligned} \rightarrow R_{\approx}(x,x) \qquad & R(z,x) \wedge R_{\approx}(x,y) \rightarrow (z,y) \\ R_{\approx}(x,y) \rightarrow R_{\approx}(y,x) \qquad & R(x,z) \wedge R_{\approx}(x,y) \rightarrow R(y,z) \\ C(x) \wedge R_{\approx}(x,y) \rightarrow C(y) \qquad & R_{\approx}(x,y) \wedge R_{\approx}(y,z) \rightarrow R_{\approx}(x,z) \end{aligned}$$

  3. For all rules $B \rightarrow H \in RB$ and $R(x, y) \in H$ with $R \in N_R^S$ simple, $\overline{P}$(RB) contains a rule $B' \rightarrow Self_R(x) \in \overline{P}$(RB), where B' is obtained from B by replacing all occurrences of y with x, all occurrences of $\{a\}(t)$ by $C_a(t)$, and (finally) all expressions $S(x, x)$ with $Self_R(x)$
  4. For each $R \in N_R^S$ and $a \in N_I$, the program $\overline{P}$(RB) contains the rule $C_a(x) \wedge R(x, x) \rightarrow Self_R(x)$

  where the role name $R_{\approx}$ is the equality predicate.

- Furthermore, given an $\mathcal{EL}^{++}$ rule base RB in normal form, RB is unsatisfiable iff $\overline{P}$(RB) is unsatisfiable. The proof for this statement is not in the scope of this work and can be checked in [36]. One can now see that the unsatisfiable problem is reduced to check unsatisfiable in a Datalog program that can be check in polynomial time.

- Satisfiability of any ELP$_n$ rule base RB is decided in time polynomial in the size of RB and exponential in n.

  More precisely, RB is transformed into an equisatisfiable Datalog program P(KB) which contains at most $max(3, n)$ variables per rule, and this transformation is possible in polynomial time in the size of RB. Furthermore, for any $C \in N_C$, $R \in N_R$, and $a, b \in N_I$, we find that:

  - RB $\models C(a)$ iff P(RB) $\models C(a)$
  - RB $\models \{a\}(b)$ iff P(RB) $\models C_a(b)$
  - RB $\models R(a, b)$ iff P(RB) $\models R(a, b)$

4. Nonmonotonic Approaches

Contrary to monotonic approaches, nonmonotonic approaches are able to perform both open and closed world reasoning over hybrid knowledge bases, as well as defining exceptions for DL concepts. This way, nonmonotonic approaches are usually concerned with defining intuitive and appropriate semantics in order to support the integration of the two components. However, by allowing closed world reasoning, the decidability problem becomes even harder, and it is necessary to define procedures that constrain the available constructors, in order to obtain a safe interaction.

To illustrate the expressivity power of integrating rules and ontologies in a nonmonotonic approach, next we recall and extend Example 3.1 from Chapter 3.

Example 4.1. *Remember the simple example presented earlier of a Thai Restaurant. In this case, we used Description Logics to model characteristics of Thai dishes, stating that Thai Curry is a Thai Dish and that every Thai Curry contains peanut oil. Moreover, we also added an ABox expressing that Sebastian ordered a Thai Curry dish. Now we can add an axiom stating that pork satay is also a Thai dish and that an unhappy costumer does not leave tip.*

$$thaiCurry \sqsubseteq ThaiDish \tag{4.1}$$
$$ThaiDish \sqsubseteq Dish \tag{4.2}$$
$$thaiCurry \sqsubseteq \exists contains.\{peanutOil\} \tag{4.3}$$
$$T \sqsubseteq \forall orderedDish.Dish \tag{4.4}$$
$$sebastian : \exists orderedDish.thaiCurry \tag{4.5}$$
$$porkSatay \sqsubseteq ThaiDish \tag{4.6}$$
$$Unhappy \sqsubseteq \neg Tip \tag{4.7}$$

At this moment, besides the previous rules monotonic rules from Example 3.1, we can add in the rule component some nonmonotonic rules.

$$dislikes(x, y) \leftarrow nutAllergic(x), nutProduct(y). \tag{4.8}$$
$$unhappy(x) \leftarrow orderedDish(x, y), dislikes(x, y). \tag{4.9}$$
$$nutAllergic(sebastian). \tag{4.10}$$
$$nutProduct(peanutOil). \tag{4.11}$$
$$contains(x, peanutOil) \leftarrow ThaiDish(x), \mathbf{not}\ nutFree(x). \tag{4.12}$$
$$nutAllergic(mark). \tag{4.13}$$
$$orderedDish(mark, porkSatay). \tag{4.14}$$

So, we defined a new nonmonotonic predicate $nutFree$ *in order to state that* by default *all Thai dishes contain peanut oil. As a consequence, since Mark is also nut allergic and ordered a Thai*

dish, it will be inferred that Mark is unhappy and that he will leave no tip. However, if afterwards we add to the knowledge base – $nutFree(porkSatay)$*, then neither* $Unhappy(mark)$ *nor* $\neg Tip(mark)$ *will hold.*

We can see that this example represents a typical nonmonotonic reasoning, as adding new information to our knowledge base ($nutFree(porkSatay)$*) falsifies what was inferred previously (*$Unhappy(mark)$ *and* $\neg Tip(mark)$*). Such default rule like* 4.12 *could never be expressed previously by any monotonic approach presented previously in Chapter 3*

Continuing along this chapter we subdivide these nonmonotonic approaches in two main categories – hybrid and homogeneous – related to how flexible the integration between rules and ontologies is made possible.

4.1 Hybrid integration

In a hybrid integration, rules are built on top of ontologies and act like restrictions that are made over ontologies. In this sense, the combination between ontologies and rules is defined over a strict semantic separation as they are treated as separated and independent components. In order to assure decidability, the connection between both is performed through a *safe interface*, which restricts the exchange of knowledge between the two sides. Ontologies are thus dealt as external sources of knowledge that provide input to rules as an external oracle. On the other hand, rules usually do not provide input to ontologies and, if they do, this input resumes to assertions on ABoxes.

Regarding the expressivity of such solutions, recall the nonmontonic example presented in the beginning of this chapter – Example 4.1. Such example cannot be fully expressed in these approaches as their semantics does not support the ontology to define an axiom which depends of a predicate from the rules component. In fact, since the ontology is seen as black box, the rules may define a predicate that is stated in the ontology, but the reverse is not possible. As a result, it is not permitted to define an axiom like $\neg Tip$ from $Unhappy \sqsubseteq \neg Tip$ which directly depends on the predicate $unhappy$ that is defined in the rules as $unhappy(x) \leftarrow orderedDish(x, y), dislikes(x, y)$.

4.1.1 HEX Programs - dlv-hex

As a hybrid approach, HEX programs deal with ontologies as external sources that can be accessed by rules that also may provide input to the ontologies. The DL knowledge base presents itself as a "black box" to the logic program. The user does not need to know its entire signature, but rather a subset of its concepts and roles, in order to extend and query with them. This strict separation avoids decidability issues that come along with a tighter integration of such diverse formalisms [52].

HEX Programs [15] consist of an extension of the stable models semantics with higher-order and external atoms. A higher-order atom allows one to quantify values over predicate names,

and to freely exchange predicate symbols with constant symbols, like in the rule:

$$C(X) \leftarrow subClassOf(D, C), D(X).$$

An external atom facilitates to determine the truth value of an atom through an external source of computation. For instance, the rule:

$$t(Sub, Pred, Obj) \leftarrow \&RDF[in](Sub, Pred, Obj), uri(in)$$

computes the predicate t taking values from the predicate $\&RDF$. The latter predicate extracts RDF statements from the set of URI specified by means of in. The truth of an external atom is delegated to an external computational source.

These external atoms empower the interaction with ontologies, allowing a bidirectional flow of information to and from external sources of computation, such as, description logic reasoners.

Formally, HEX Programs are built on mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of *constant names, variable names* and *external predicate names*, respectively. Elements from $\mathcal{X}$ (resp. $\mathcal{C}$) are denoted with first letter in upper case (resp. lower case), and elements from $\mathcal{G}$ are prefixed with "&". Constant names serve both as individual and predicate names. $\mathcal{C}$ may be infinite. Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*.

An external atom is of the form $\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$, where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two list of terms (called input and output list, respectively), and $\&g$ is an external predicate name. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates. A higher-order atom is a tuple $Y_0(Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms and $n \geq 0$ is the arity of the atom.

A HEX Program is a finite set P of rules of the form [10]:

$$\alpha_1 \vee \ldots \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_m \qquad (m, n, k \geq 0)$$

where $\alpha_1, \ldots, \alpha_k$ are higher-order atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms. The operator not represents negation as failure. $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^- = \{\beta_n + 1, \ldots, \beta_m\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*.

The semantics of HEX programs is given by a generalisation of the stable models semantics by FLP-*reduct* [4], and consequently, query answering may yield none, one, or multiple models.

The *Herbrand base* of a HEX program P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule r, $ground(r)$, is defined accordingly, and the grounding of a program P is $ground(P) = \bigcup_{r \in P} ground(r)$.

An *interpretation relative* to P is any subset $I \subseteq HB_P$ containing only atoms. I is said to be a *model* of a atom $a \in HB_P$, denoted $I \models a$, if $a \in I$. With every external predicate name $\&g \in \mathcal{G}$ it is associate an $(n+m+1)$-ary Boolean function $f_{\&g}$ (called oracle function) assigning each tuple $(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where $n = in(\&g)$, $m = out(\&g)$, $I \subseteq HB_P$ and

$x_i, y_j \in \mathcal{C}$. $I \subseteq HB_P$ is a *model* of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, iff $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$.

Let r be a ground rule:

- $I \models H(R)$ iff there is some $a \in H(r)$ such that $I \models a$,
- $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$,
- $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$.

I is a *model* of a HEX program P, denoted $I \models P$, iff $I \models r$ for all $r \in ground(P)$. P is satisfiable if it has some model.

Given a HEX program P, the FLP-*reduct* of P with respect to $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in ground(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is a *stable model* of P iff I is a minimal model of fP^I.

dlvhex[1] is a prototype implementation of a solver for HEX programs, reusing and integrating existing reasoning applications instead of writing them from scratch. Thus, **dlvhex** can be seen as a reasoner framework rather than as a stand-alone inference engine [52].

As a computational aspect, recalling that NEXP denotes nondeterministic exponential time, and C^D, for complexity classes C and D, denotes complexity in C with an oracle for a problem in D. Suppose that for every $\&g \in \mathcal{G}$, then the function $f_{\&g}$ has complexity in C. Then, deciding whether a HEX program P has some stable model has the complexity of NEXP^{NP^C}.

4.2 Homogeneous integration

The homogeneous integration tries to overcome the expressive limitations from the previous combinations by completely eliminating the separation between ontologies and rules. They aim at an intuitive and full combination of DL formulas and rules without restrictions, with a unique vocabulary where predicates can be defined either using rules or using DL.

Once again, recall Example 4.1 defined earlier. Contrary to nonmonotonic approaches that only accomplish an hybrid integration, solutions that we define here as homogeneous are able to fully express such an example, since every predicate or axiom can be stated in any of the components in an unrestricted way. Intuitively, each solution that we present along the remaining section diverge from the others by its semantics and by complexity of its reasoning algorithms.

However, for these approaches, it is still not possible to perform such homogeneous combination without compromising decidability, which led to a concept named *DL-safe* that is explained next.

[1] `http://www.kr.tuwien.ac.at/research/dlvhex/`

4.2.1 DL-Safeness

As shown in Section 3.2, combining arbitrary first-order rules with decidable description logics, even the ones who contain just the very basic DL constructs, easily leads to undecidable of the the satisfiability problem.

One way of preserving decidability is to restrict this combination for rules called *safe*.

Definition 4.1. A DL rule r is *DL-safe* if each variable occurring in r also occurs in a non-DL-atom in the body of r. A program P is DL-safe if all its rules are DL-safe.

Roughly speaking, a safe rule is a rule which is only applicable to known individuals, explicitly introduced in the ABox. In practice, DL-safeness is obtained by adding an assertion $O(\alpha)$ for each known object α. For example [44], if $Person$, $livesAt$, and $worksAt$ are concepts and roles from the knowledge base, the following rule is not DL-safe:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y)$$

because the variables x and y occur in DL-atoms, but do not occur in a body atom with a predicate outside of the knowledge base. This rule can be made DL-safe by adding $O(x)$ and $O(y)$ to the body of the rule:

$$Homeworker(x) \leftarrow Person(x), livesAt(x, y), worksAt(x, y), O(x), O(y)$$

DL-safety restricts the interchange of consequences between the component languages to those consequences involving individuals explicitly introduced in the ABox. The impact of this DL-safety restriction depends on the type of application used. For applications relying mainly on extensional reasoning (such as metadata-based information retrieval), DL-safety does not represent a serious restriction as the universe is usually limited to known objects. On the contrary, in applications requiring intensional reasoning (such as natural language processing), DL-safety has much more severe restrictions, as many conclusions drawn involve unnamed objects [44].

4.2.2 $\mathcal{DL} + log$

$\mathcal{DL} + log$ [50] is a general framework for the integration of Description Logics and disjunctive Datalog which tries to overcome some of the limitations of DL-safeness condition through a new safeness condition named *weak safeness*. This new condition realises a tighter form of interaction between DL-KBs and rules, increasing the expressive power: conjunctive queries (and unions of conjunctive queries) can be expressed in $\mathcal{DL} + log$.

Simultaneously, $\mathcal{DL} + log$ is still decidable in many DLs by exploiting the deep relationship between query containment in DLs and reasoning in $\mathcal{DL} + log$.

4.2.2.1 Syntax

Next, we recall the $\mathcal{DL} + log$ syntax, which constitutes an extension of $\mathcal{DL} - log$ originally proposed in [49].

$\mathcal{DL} + log$ syntax defines three mutually disjoint predicate alphabets:

- an alphabet of concept names Σ_C;
- an alphabet of role names Σ_R;
- an alphabet of *Datalog* predicates Σ_D

An *atom* is an expression of the form $p(X)$, where p is a predicate of arity n and X is a n-tuple of variables and constants. If no variable symbol occurs in X, then $p(X)$ is called *ground atom* (or fact). If $p \in \Sigma_C \cup \Sigma_R$ the atom is called a DL-atom, while if $p \in \Sigma_D$, it is called a Datalog atom.

A Datalog$^{\neg\vee}$ rule R is an expression of the form:

$$p_1(X_1) \vee \cdots \vee p_n(X_n) \leftarrow r_1(Y_1), \ldots, r_m(Y_m), \mathbf{not}\ u_1(W_1), \ldots, \mathbf{not}\ u_h(W_h)$$

such that $n \geq 0,\ m \geq 0,\ h \geq 0$, each $p_i(X_i),\ r_i(Y_i),\ u_i(W_i)$ is an atom. The variables occurring in the atoms $p_1(X_1), \ldots, p_n(X_n)$ are called the *head variables* of R. If $n = 0$, R is called a *constraint*. A Datalog$^{\neg\vee}$ program is a set of Datalog$^{\neg\vee}$ rules. If for all $R \in \mathcal{P},\ n \leq 1,\ \mathcal{P}$ is called a Datalog$^{\neg\vee}$ program. If, for all $R \in \mathcal{P},\ h = 0,\ \mathcal{P}$ is called a positive disjunctive Datalog program. If, for all $R \in \mathcal{P},\ n \leq 1$ and $h = 0,\ \mathcal{P}$ is called a positive Datalog program. If there are no occurrences of variable symbols in $\mathcal{P}$, $\mathcal{P}$ is called a ground program.

Definition 4.2. Given a description logic $\mathcal{DL}$, a $\mathcal{DL}$-knowledge base with weakly-safe Datalog$^{\neg\vee}$ rules $\mathcal{B}$ is a pair $(\mathcal{K}, \mathcal{B})$, where:

- $\mathcal{K}$ is a $\mathcal{DL}$-KB, i.e., a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ is the ABox;
- $\mathcal{P}$ is a set of Datalog$^{\neg\vee}$ rules, where each rule R has the form:

$$p_1(X_1) \vee \cdots \vee p_n(X_n) \leftarrow r_1(Y_1), \ldots, r_m(Y_m), s_1(Z_1), \ldots, s_k(Z_k), \mathbf{not}\ u_1(W_1), \ldots, \mathbf{not}\ u_h(W_h) \tag{4.15}$$

  where $n \geq 0,\ m \geq 0,\ k \geq 0\ h \geq 0$, each $p_i(X_i),\ r_i(Y_i),\ s_i(Z_i),\ u_i(W_i)$ is an atom and:

  - each p_i is either a DL-predicate or a Datalog predicate;
  - each $r_i,\ u_i$ is a Datalog predicate:
  - each s_i is a DL-predicate
  - (Datalog safeness) every variable occurring in R must appear in at least one of the atoms $r_1(Y_i), \ldots, r_m(Y_m),\ s_1(Z_1), \ldots, s_k(Z_k)$;

– (weak safeness) every head variable of R must appear in at least one of the atoms $r_1(Y_1), \ldots, r_m(Y_m)$.

In $\mathcal{DL}+log$ with weak safety it is possible to access unnamed individuals in classical atoms, something which is not possible with the ordinary DL-Safe condition.

In this approach, the standard names assumption is employed in the definition of the semantics: the interpretation Δ corresponds one-to-one to a countable infinite set of constants $\mathcal{C}$ of the signature.

$\mathcal{DL}+log$ comes with two types of semantics and exercises reasoning through stable models. The first-order semantics is obtained by interpreting each rule of the form (4.15) as the following first-order implication, where x is the set of free variables of the rule:

$$\forall x : p_1 \vee \cdots \vee p_n \subset r_1 \wedge \cdots \wedge r_m \wedge s_1 \wedge \ldots s_k \wedge \neg u_1 \wedge \cdots \wedge \neg u_h \tag{4.16}$$

For the definition of the nonmonotonic semantics, the following standard definitions for datalog programs are needed. For $\mathcal{P}_G$, a ground Datalog program, and I an interpretation, the GL-reduct of $\mathcal{P}_G$ with respect to I, written $\mathsf{GL}(\mathcal{P}_G, I)$ is obtained by transforming each rule $R \in \mathcal{P}_G$ as follows:

- Delete R if it contains a negated atom $\mathbf{not}\ B_i$ such that $I \models B_i$;
- Delete all negated body atoms $\mathbf{not}\ B_i$ such that $I \not\models B_i$

An interpretation I is a *model* of a ground datalog program $\mathcal{P}_G$ without $\mathbf{not}$-atoms if it satisfies the rules of $\mathcal{P}_G$ when these are interpreted as implications (4.16). I is a *minimal model* if no interpretation $I' \subset I$ is a model of $\mathcal{P}_G$. For a ground datalog program $\mathcal{P}_G$ possibly containing $\mathbf{not}$-atoms, an interpretation I is a *stable model* if it is the minimal model of $\mathsf{GL}(\mathcal{P}_G, I)$.

It is now possible to define the nonmonotonic semantics of $\mathcal{DL} + log$. Let $\mathsf{gr}(\mathcal{P})$ be the ground program obtained by replacing in each rule from $\mathcal{P}$ all variables with constants from Δ in all possible ways. For an interpretation I and a set of predicates Σ, I_Σ is the interpretation obtained by restricting I to the predicates in Σ. Furthermore, for $\mathcal{P}_G$ a ground program, $\Pi(\mathcal{P}_G, I)$ is the *projection of $\mathcal{P}_G$ with respect to I* and Σ is equal to the set of rules obtained by transforming each rule $R \in \mathcal{P}_G$ as follows:

- Delete R if a head atom H_i with a predicate from Σ exists in R such that $I \models H_i$;
- Delete each head atom H_i with a predicate from Σ if $I \not\models H_i$;
- Delete R if a body atom B_i with a predicate from Σ exists in R such that $I \not\models B_i$;
- Delete each body atom B_i with a predicate from Σ if $I \models H_i$.

I is a nonmonotonic model of a $\mathcal{DL} + log$ knowledge base $\mathcal{K}$ if it is a FOL model of $\mathcal{K}$ and I_{Σ_D} is a stable model of $\Pi(\mathsf{gr}(\mathcal{P}), I_{\Sigma_C \cup \Sigma_R})$.

4.2.3 MKNF

MKNF [43], Minimal Knowledge and Negation as Failure, strongly related to MBNF [38], represents one of the most robust proposals made so far to integrate open and closed world reasoning using the style of logic programming combined with DL knowledge bases.

It evaluates hybrid and homogeneous knowledge bases under a stable model semantics with the support of *DL-safety*.

MKNF is an extension of first-order logic with two modal operators **K** and **not**, which support closed world reasoning. The first is an epistemic operator and can be read as “is known", while the second represents the negation by default.

In a similar way to *DL+log*, MKNF uses Standard Name Assumption, SNA, instead of the standard Unique Name Assumption, UNA, present in nonmonotonic formalisms. Since DLs do not assume UNA, the idea behind such concept is to provide a method that allows closed world reasoning and simultaneously enable the DL to define synonyms. Hence, the authors consider $\approx$ as a congruent relation which makes the DL to assume that two individuals are the same. Roughly speaking, this approach assume UNA at the level of logic programming to ensure that **not** has an intuitive semantics; however, at the level of DL, $\approx$ is used to explicitly state equality between individuals. Therefore, two individuals are only assumed to be equal if there is an explicit evidence for doing so.

As an illustration, consider a small example scenario from [34].

Example 4.2. *Consider an online store selling, among other things, CDs. Due to the fact that many newly published CDs are simply compilations of already existing music, the owners decide to offer their customers a special service: whenever somebody likes the compilation of a certain artist he can search for more music of that artist published on albums. The service shall however deny offering other compilations or products which are too similar to the already owned CD. This similarly can be defined in various ways, but it is assumed for simplicity that this is handled internally, e.g. by counting the number of identical tracks, and encoded by predicate Dif(x, y). The internal database is organised as a hybrid MKNF knowledge base including an ontology containing all available discs, their tracks and so on and whether they are albums or compilations. The following shall provide the considered service:*

$$\mathit{Comp} \sqsubseteq \neg\, \mathit{Offer} \tag{4.17}$$

$$\mathbf{K}\ \mathit{Offer}(x) \leftarrow \mathbf{not}\ \mathit{owns}(x), \mathbf{K}\ \mathit{owns}(y), \mathbf{K}\ \mathit{Dif}(x, y), \mathbf{K}\ \mathit{artist}(x, z), \mathbf{K}\ \mathit{artist}(y, z). \tag{4.18}$$

Given the input of CDs the customer owns, rule (4.18) *offers an album x in case the customer does not own it, which is sufficiently different to a CD y he owns, where the artist z of x is the same as the artist of y. Additionally,* (4.17) *is a DL statement (translatable into $\forall x$: Comp$(x) \rightarrow \neg$ Offer(x)) enforcing that any CD which is a compilation shall never be offered.*

MKNF employs several algorithms to check entailment for different classes of hybrid knowledge bases. They consider a variety of cases, when the rules are positive, positive nondisjunctive, stratified nondisjunctive and nonstratified nondisjuctive, where each case has the corresponding complexity. Nevertheless, its basic idea consists in generating several possible models

and choose the minimal one.

4.2.3.1 MKNF notions

Next, we recall important notions from [43]. A hybrid MKNF knowledge base $\mathcal{K}$ consists of a knowledge base $\mathcal{O}$ in any decidable description logic $\mathcal{DL}$ and a set $\mathcal{P}$ of MKNF *rules* of the following form:

$$\mathbf{K}H_1 \vee ... \vee \mathbf{K}H_n \leftarrow \mathbf{K}B_1^+, ..., \mathbf{K}B_m^+, \mathbf{not}B_1^-, ..., \mathbf{not}B_k^- \tag{4.19}$$

The sets $\{\mathbf{K}H_i\}$,$\{\mathbf{K}B_i^+\}\{\mathbf{K}B_i^-\}$ are called the *rule head*, the *positive body* and the *negative body*, respectively. A rule r is non-disjunctive if $n = 1$; r is positive if $k = 0$; r is a *fact* if $m = k = 0$; r is *safe* if all variables in r occur in a positive body atom. A program $\mathcal{P}$ is a finite set of MKNF rules. A hybrid MKNF knowledge base $\mathcal{K}$ is a pair $(\mathcal{O}, \mathcal{P})$, where $\mathcal{O}$ represents a DL knowledge base.

As in SWRL, H_i, B_i^+, and B_i^- are first-order atoms of the form $p(t_1, .., t_n)$ where p is a predicate and the t_i are first-order terms. If φ is an MKNF formula, then $\neg\varphi$, $\exists x : \varphi$, $\mathbf{K}\varphi$ and $\mathbf{not}\varphi$ are MKNF formulas as well as $\varphi_1 \wedge \varphi_2$ for MKNF formulas φ_1, φ_2. The operators $\wedge$, $\forall$, $\leftarrow$, $\equiv$ are shortcuts for the usual boolean combinations of the previously introduced syntax. A formula of the form $\mathbf{K}\varphi$ is called a *modal* **K** -**atom**, a formula of the form $\mathbf{not}\varphi$ is called a *modal* **not**-**atom**, indiscriminately, modal **K**- and **not**-atoms are called *modal atoms*.

An occurrence of a modal atom in an MKNF formula is called *strict* if the atom does not occur in scope of a modal operator. An MKNF formula without any free variables is a *sentence* and *ground* if it does not contain variables at all. φ is *modally closed* if all modal operators are applied only to sentences. φ is *positive* if it does not contain the operator **not**. φ is *subjective* if all first-order atoms of φ occur within the scope of a modal operator. φ is *flat* if all occurrences of modal atoms in φ are strict and φ is subjective. φ is *objective* if it does not contain modal operators. Substituting the free variables x_i in φ by terms t_i is denoted $\varphi[t_1/x_1, ..., t_n/x_n]$.

Recalling from [43], by the employment of *standard names assumption* it is assumed that, apart from the constants occurring in the formulae, the signature contains a countably infinite supply of constants not occurring in the formulae. The Herbrand Universe of such signature is also called *domain set* Δ. An MKNF *structure* is a triple (I, M, N) where I is an Herbrand first-order interpretation over Δ and the equality predicate $\approx$ is interpreted in I and in each interpretation from M and N as a congruence relation, i.e., it is reflexive, symmetric, transitive, and it allows one to replace equals by equals. Satisfiability of MKNF sentences in an MKNF structures (I, M, N) is defined as follows:

$$\begin{array}{ll}
(I, M, N) \models p(t_1, ..., t_n) & \text{iff } p(t_1, ..., t_n) \in I \\
(I, M, N) \models \neg\varphi & \text{iff } (I, M, N) \not\models \varphi \\
(I, M, N) \models \varphi_1 \wedge \varphi_2 & \text{iff } (I, M, N) \models \varphi \text{ and } (I, M, N) \models \varphi_2 \\
(I, M, N) \models \exists x : \varphi & \text{iff } (I, M, N) \models \varphi[\alpha/x] \text{ for some } \alpha \in \Delta \\
(I, M, N) \models \mathbf{K}\,\varphi & \text{iff } (J, M, N) \models \varphi \text{ for all } J \in M \\
(I, M, N) \models \mathbf{not}\,\varphi & \text{iff } (J, M, N) \not\models \varphi \text{ for some } J \in N
\end{array}$$

An MKNF *interpretation* M is a non-empty set of Herbrand first-order interpretations over Δ and *models* a closed MKNF formula φ, i.e. $M \models \varphi$, if $(I, M, M) \models \varphi$ for each $I \in M$. An MKNF interpretation M is an MKNF *model* of a closed MKNF formula φ if M models φ and for each MKNF interpretation M' such that $M' \supset M$ and $(I', M', M) \not\models \varphi$ for some $I' \in M'$.

An MKNF formula φ is *MKNF satisfiable* if an MKNF model of φ exists and *MKNF unsatisfiable* otherwise. Given MKNF formulas φ and ψ, φ *MKNF entails* ψ, written $\varphi \models_{MKNF} \psi$, if $(I, M, M) \models \psi$ for each MKNF model M of φ and $I \in M$.

The semantics of $\mathcal{K}$ is obtained by translating it into a first-order MKNF formula as follows:

Definition 4.3. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a hybrid MKNF knowledge base. π is an extension of r, $\mathcal{P}$, and $\mathcal{K}$ as follows, where x is the vector of the free variables of r:

$$\pi(r) = \forall x : (\mathbf{K}H_1 \wedge ... \wedge \mathbf{K}H_n \subset \mathbf{K}B_1^+ \wedge ... \wedge \mathbf{K}B_m^+ \wedge \mathbf{not}B_1^- \wedge ... \wedge \mathbf{not}B_k^-)$$

$$\pi(\mathcal{P}) = \bigwedge_{r \in \mathcal{P}} \pi(r)$$

$$\pi(\mathcal{K}) = \mathbf{K}\pi(\mathcal{O}) \wedge \pi(\mathcal{P})$$

A hybrid MKNF knowledge base $\mathcal{K}$ is satisfiable if and only if an MKNF model of $\pi\mathcal{K}$ exists. Furthermore, $\mathcal{K}$ entails an MKNF formula ψ, written $\mathcal{K} \models \psi$, if and only if $\pi(\mathcal{K}) \models_{MKNF} \psi$.

Definition 4.4. An MKNF rule r is DL-safe if every variable in r occurs in at least one non-DL-atom $\mathbf{K}$ B occurring in the body of r. A hybrid MKNF knowledge base $\mathcal{K}$ is DL-safe if all its rules are DL-safe.

In practice, DL-safeness is obtained by adding an assertion $O(\alpha)$ for each known object α and then adding to the body of each MKNF rule r a non-DL atom $\mathbf{K}O(x)$ for all variables x occurring in r.

Given a hybrid MKNF knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{P})$, the *ground instantiation* of $\mathcal{K}$ is the KB $\mathcal{K}_G = (\mathcal{O}, \mathcal{P}_G)$ where $\mathcal{P}_G$ is obtained by replacing in each rule of $\mathcal{P}$ all variables with constants from $\mathcal{K}$ in all possible ways. For a DL-safe hybrid KB $\mathcal{K}$ and a ground MKNF formula ψ, $\mathcal{K} \models \psi$, if and only if $\mathcal{K}_G \models \psi$.

4.2.3.2 Reasoning Algorithms

In [43] five different algorithms are presented for reasoning with different types of modally closed MKNF formulae, as all these cases differ in the complexity of reasoning. The first algorithm handles flat MKNF formulae, the second handles positive MKNF formulae, the third handles positive non-disjunctive MKNF formulae, the fourth handles stratified non-disjunctive MKNF formulae, and finally, the fifth handles non-stratified non-disjunctive MKNF formulae.

Given a non-ground hybrid MKNF knowledge base $\mathcal{K}$, all these algorithms first compute the ground knowledge base $\mathcal{K}_G$ to obtain an MKNF theory without modal operators under quantifiers.

Satisfiability of some formula is usually demonstrated by constructing a model for the formula. However, since an MKNF model M of an MKNF formula σ is an infinite set of first-order interpretations, for a practical algorithm, an appropriate finite representation of M is needed. This way, in [43] there is no direct representation of M, but rather a first order formula φ such that M is exactly the set of first-order models of φ; this is usually written as $M = \{I | I \models \varphi\}$. The formula φ is uniquely defined through a partition (P,N) of modal atoms of σ into positive and negative ones and it corresponds to the *objective knowledge*, written ob_P which can be computed from the atoms chosen to be positive in a straightforward way. These can be formalised as follows:

Definition 4.5. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a hybrid MKNF knowledge base. The set of $\mathbf{K}$-atoms of $\mathcal{K}$, written $\mathsf{KA}(\mathcal{K})$, is the smallest set that contains:

1. all $\mathbf{K}$-atoms of $\mathcal{P}_G$,
2. a modal atom $\mathbf{K}\xi$ for each modal atom $\mathbf{not}\xi$ occurring in $\mathcal{P}_G$.

For a subset P of $\mathsf{KA}(\mathcal{K})$, the objective knowledge of P is the formula $\mathsf{ob}_{\mathcal{K},P} = \mathcal{O} \cup \bigcup_{\mathbf{K}\xi \in P} \xi$. A partition (P, N) of $\mathsf{KA}(\mathcal{K})$ is consistent if $\mathsf{ob}_{\mathcal{K},P} \not\models \xi$ for each $\mathbf{K}\xi \in N$.

Let φ be an MKNF formula and (P, N) a partition of $\mathsf{KA}(\mathcal{K})$. The formula $\varphi[\mathbf{K}, P, N]$ is obtained from φ by replacing each $\mathbf{K}\xi$ with `true` if $\mathbf{K}\xi \in P$ and with `false` otherwise. $\varphi[\mathbf{not}, P, N]$ is obtained from φ by replacing each $\mathbf{not}\xi$ with `true` if $\mathbf{K}\xi \in N$ and with `false` otherwise. Finally, $\varphi[P, N] = \varphi[\mathbf{K}, P, N][\mathbf{not}, P, N]$.

For a set of modal atoms S, S_{DL} is the subset of DL-atoms of S,$\widehat{S} = \{\xi | \mathbf{K}\xi \in S\}$, and $\widehat{S}_{DL} = \widehat{S'}$ for $S' = S_{DL}$.

Next, it we present the different strategies for computing (P, N).

- **The General Case.** As for disjunctive datalog, in the general case it is necessary to guess a partition (P, N) of $\mathsf{KA}(\mathcal{K})$. This is captured by the algorithm in Figure 4.1. Here it is applied the known technique from LP - stable models.

  Assuming that the satisfiability of first-order formulae in Figure 4.1 is decidable with complexity $\mathcal{C}$, the complexity of the algorithm not-entails-DL(σ, ψ) is in $\mathcal{E}^{\mathcal{E}}$, where $\mathcal{E} =$ NP if $\mathcal{E} \subseteq$ NP, and $\mathcal{E} = \mathcal{C}$ otherwise.

  The idea is that the main algorithm for reasoning in MKNF knowledge bases raises the complexity of the DL employed in one level. This means that, e.g., if the DL is tractable, then this algorithm runs in NP.

- **Positive Programs.** For positive queries, it is not necessary to ensure the preference semantics of MKNF. This way, the algorithm is the same as in Figure 4.1, only without Condition (5), running with data complexity $\mathcal{E}$.

Algorithm: not-entails-DL($\mathcal{K}, \psi$)

Input:
$\mathcal{K} = (\mathcal{O}, \mathcal{P})$: a DL-safe hybrid MKNF knowledge base
ψ: a ground formula $(\neg)\mathbf{K}A$
Output:
true if $\mathcal{K} \not\models \psi$; false otherwise

let $\mathcal{K}_G$ be the ground instantiation of $\mathcal{K}$
if *a partition* (P, N) *of* $\mathsf{KA}(\mathcal{K}_G) \cup \{\mathbf{K}A\}$ *exists such that*

1. $\mathcal{P}_G[P, N]$ *evaluates to* true*, and*
2. $\mathcal{O} \cup \widehat{P}_{DL}$ *is satisfiable, and*
3. $\mathcal{O} \cup \widehat{P}_{DL} \not\models \xi$ *for each* $\mathbf{K}\xi \in N_{DL}$*, and*
4. ***for each*** $Q(a_1, ..., a_n) \in \widehat{N}$ *and* $Q(b_1, ..., b_n) \in \widehat{P}$*, there is* $\mathcal{O} \cup \widehat{P}_{DL} \not\models a_i \approx b_i$*, for some* $1 \leq i \leq n$
5. ***for*** $\gamma = \mathcal{P}_G[\mathbf{not}, P, N]$ *and each partition* (P', N') *of* P *such that* $N' \neq 0$
   (a) $\gamma[P', N \cup N']$ *evaluates to* false*, or*
   (b) $\mathcal{O} \cup \widehat{P}'_{DL}$ *is unsatisfiable, or*
   (c) $\mathcal{O} \cup \widehat{P}'_{DL} \models \xi$ *for some* $\mathbf{K}\xi \in N'_{DL}$*, or*
   (d) ***for some*** $Q(a_1, ..., a_n) \in \widehat{N}'$ *and* $Q(b1, ..., b_n) \in \widehat{P}'$*, there is* $\mathcal{O} \cup \widehat{P}'_{DL} \models a_i \approx b_i$*, for all* $1 \leq i \leq n$
6. *one of the following conditions holds:*
   (a) $\psi = \mathbf{K}A$ *and* $\mathbf{K}A \notin P$*, or*
   (b) $\psi = \neg\mathbf{K}A$ *and* $\mathbf{K}A \in P$

then return true ;
else return false

Figure 4.1 Entailment in General hybrid MKNF KBs

- **Positive Nondisjunctive Programs.** If $\mathcal{K}$ is nondisjunctive and positive, (P, N) is constructed deterministically in a bottom–up fashion. Each positive DL-safe non-disjunctive hybrid MKNF knowledge base $\mathcal{K}$ has at most one MKNF model.

  Definition 4.6. For $\mathcal{K}$ a positive non-disjunctive DL-safe hybrid MKNF knowledge base, $R_\mathcal{K}, D_\mathcal{K}$, and $T_\mathcal{K}$ are the operators defined on the subsets of $\mathsf{KA}(\mathcal{K})$ as follows:

  $$R_\mathcal{K}(S) = S \cup \{\mathbf{K}H | \mathcal{K}_G \text{ contains a rule of the form (4.19) such that } \mathbf{K}B_i \in S \text{ for each } 1 \leq i \leq n\}$$

  $$D_\mathcal{K}(S) = \{\mathbf{K}\xi | \mathbf{K}\xi \in \mathsf{KA}(\mathcal{K}) \text{ and } \mathcal{O} \cup \widehat{S}_{DL} \models \xi\} \cup \{\mathbf{K}Q(b1, ..., b_n) | \mathbf{K}Q(a1, ..., a_n) \in S \backslash S_{DL} \text{ and } \mathcal{O} \cup \widehat{S}_{DL} \models s_i \approx b_i, \text{ for } 1 \leq i \leq n\}$$

  $$T_\mathcal{K}(S) = R_\mathcal{K}(S) \cup D_\mathcal{K}(S)$$

  The operator $T_\mathcal{K}$ is monotonic on the lattice of the subsets of $\mathsf{KA}(\mathcal{K})$ and has the least fixpoint denoted with $T^\omega_\mathcal{K}$. For entailment checking, $\mathcal{K} \models \mathbf{K}A$ iff $\mathsf{ob}_{\mathcal{K},T^\omega_\mathcal{K}} \models A$, and $\mathcal{K} \models \neg\mathbf{K}A$ iff $\mathsf{ob}_{\mathcal{K},T^\omega_\mathcal{K}} \not\models A$. The data complexity of computing $T^\omega_\mathcal{K}$ is in $\mathsf{P}^\mathcal{C}$.

- **Stratified Programs.** In stratified programs deriving a **K**-atom by a rule in a stratum i should not affect the values of modal atoms in strata below i.

  This means that, when evaluating a stratum σ_i, the values of all **not**-atoms in σ_i have already been computed. Hence, evaluating any subsequent stratum will not change the values of any **K**- and **not**-atoms from any stratum σ_j. For ordinary datalog programs, a stratification is defined by the strongly connected components of the dependency graph associated with the program. However, MKNF programs can contain arbitrary first-order formulae as atoms, which makes checking stratification more difficult.

  Consider the following program σ as an example from [43]:

  $$\mathbf{K}(p \vee q) \leftarrow \mathbf{not}p \tag{4.20}$$

  $$\mathbf{K}(\neg q) \leftarrow \mathbf{K}(p \vee q) \tag{4.21}$$

  The dependency graph of σ would suggest a stratification in which (4.20) comes before (4.21). In fact, evaluating the rules in this order, the body of (4.20) is satisfied and $\mathbf{K}(p \vee q)$ is derived. This satisfies the body of (4.21), and so $\mathbf{K}(\neg q)$ is derived as well. Thus, the objective knowledge is now $(p \vee q) \wedge \neg q$, which is equivalent to p. However, the body of (4.20) is not satisfied any more, so the model $M = \{I | I \models p\}$ is not minimal. This program is not stratified as the evaluation of (4.21) changed the valued already computed for (4.20). In fact, Algorithm 4.1 shows that σ has no MKNF models.

  To compute models for stratified MKNF program σ [43] presents an algorithm that processes strata sequentially:

Definition 4.7. For a stratification $\sigma_1, \ldots, \sigma_k$ of an MKNF program σ, the sequence of subsets $U_0, \ldots, U_k$ of $\mathsf{KA}(\sigma)$ is inductively defined as $U_0 = \emptyset$ and, for $0 < i \leq k$, $U_i = T^{\omega}_{\mathcal{X}i}$, where $\mathcal{X}i = U_{i-1} \cup \sigma'_i$ and σ'_i is obtained from σ_i by replacing each atom $\mathbf{not}\xi$ with its value in U_{i-1}. Finally, $U^{\omega}_{\sigma} = U_k$.

The partition $(U^{\omega}_{\sigma}, \mathsf{KA}(\sigma) \backslash U^{\omega}_{\sigma})$ induces the MKNF model of a stratified program σ. The data complexity of computing $U^{\omega}_{\mathcal{K}}$ is in $\mathsf{P}^{\mathcal{K}}$.

- **Non-disjunctive Non-stratified Programs.** For the case when σ is a non-disjunctive program and a stratification cannot be found, one still needs to guess the partition (P, N) to determine the objective knowledge. However, $\sigma[\mathbf{not}, P, N]$ is a positive non-disjunctive MKNF program, which is possible to apply 4.6. Thus, the algorithm used is the same as Algorithm 4.1, but replacing Condition (5) with $T^{\omega}_{\gamma} = P$, running with data complexity $\mathcal{E}^{\mathsf{P}^{\mathcal{C}}}$.

4.2.4 3-Valued MKNF Semantics

The 3-valued MKNF semantics [34] defines a semantics that relates to MKNF, in the same way as well-founded is related to stable models semantics.

As described earlier, in the previous proposal, even with polynomial DL, the addition of rules turns reasoning in coNP. So, in order to overcome these limitations of stable models, the main goal here is to provide a more efficient reasoning for MKNF by benefiting from the characteristics of the well-founded semantics. Its major advantage relies on the definition of simpler and lighter methods for calculating a model, which incorporates only the minimal necessary true information.

Now, satisfiability is no longer defined as in [43], where modal atoms are allowed only to be either true or false in a given MKNF structure. In [34] this framework is extended with a third value **u**, denoting *undefined*, enabling the computational of the well-founded model of an MKNF program that is a sound approximation of stable models.

The undefined value is only assigned to modal atoms; first-order atoms remain 2-valued due to being interpreted solely in one first-order interpretation. Hence, a rule free hybrid knowledge base shall be interpreted just as any DL base. The MKNF structure is extended as follows.

Definition 4.8. A 3-valued MKNF structure $(\mathcal{I}, \mathcal{M}, \mathcal{N})$ consists of a Herbrand first-order interpretation I and two pairs $\mathcal{M} =< M, M_1 >$ and $\mathcal{N} =< N, N_1 >$ of sets of Herbrand first-order interpretations where any first-order atom which occurs in all elements in M, N respectively, also occurs in all elements of M_1, respectively N_1. It is called total if $\mathcal{M} =< M, M >$ and $\mathcal{N} =< N, N >$.

The evaluation of MKNF sentences is now defined with respect to the set {**t,u,f**} of truth values with the given order $\mathbf{f} < \mathbf{u} < \mathbf{t}$:

- $(\mathcal{I},\mathcal{M},\mathcal{N})(p(t_1,\ldots,t_n)) = \begin{cases} \mathbf{t} & \text{iff } p(t_1,\ldots,t_n) \in I \\ \mathbf{f} & \text{iff } p(t_1,\ldots,t_n) \notin I \end{cases}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\neg\varphi) = \begin{cases} \mathbf{t} & \text{iff } (\mathcal{I},\mathcal{M},\mathcal{N})(\varphi) = \mathbf{f} \\ \mathbf{u} & \text{iff } (\mathcal{I},\mathcal{M},\mathcal{N})(\varphi) = \mathbf{u} \\ \mathbf{f} & \text{iff } (\mathcal{I},\mathcal{M},\mathcal{N})(\varphi) = \mathbf{t} \end{cases}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\varphi_1 \wedge \varphi_2) = min\{(\mathcal{I},\mathcal{M},\mathcal{N})(\varphi_1),(\mathcal{I},\mathcal{M},\mathcal{N})(\varphi_2)\}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\varphi_1 \supset \varphi_2) = \begin{cases} \mathbf{t} & \text{iff } (\mathcal{I},\mathcal{M},\mathcal{N})(\varphi_1 \geq (\mathcal{I},\mathcal{M},\mathcal{N}) \\ \mathbf{f} & \text{otherwise} \end{cases}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\exists x : \varphi) = max\{(\mathcal{I},\mathcal{M},\mathcal{N})(\varphi[\alpha/x])|\alpha \in \Delta\}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\mathbf{K}\,\varphi) = \begin{cases} \mathbf{t} & \text{iff } (J,< M,M_1 >,\mathcal{N})(\varphi) = \mathbf{t} \text{ for all } J \in M \\ \mathbf{f} & \text{iff } (J,< M,M_1 >,\mathcal{N})(\varphi) = \mathbf{t} \text{ for some } J \in M_1 \\ \mathbf{u} & \text{otherwise} \end{cases}$
- $(\mathcal{I},\mathcal{M},\mathcal{N})(\mathbf{not}\,\varphi) = \begin{cases} \mathbf{t} & \text{iff } (J,\mathcal{M},< N,N_1 >)(\varphi) = \mathbf{f} \text{ for some } J \in N_1 \\ \mathbf{f} & \text{iff } (J,\mathcal{M},< N,N_1 >)(\varphi) = \mathbf{t} \text{ for all } J \in N \\ \mathbf{u} & \text{otherwise} \end{cases}$

This approach also extends the MKNF interpretations, since one set of first-order interpretations is not sufficient to represent truth, falsity, and undefinedness of modal atoms.

Definition 4.9. An interpretation pair (M,N) consists of two MKNF interpretations M, N with $N \subseteq M$, and models a closed MKNF formula φ, written $(M,N) \models \varphi$, if and only if $(I,< M,N >,< M,N >)(\varphi) = \mathbf{t}$ for each $I \in M$. If there exists an interpretation pair modelling φ, then φ is consistent.

M contains all interpretations which model only truth while N models everything which is true or undefined. Evidently, just as in the two-valued case, anything not being modeled in N is false by default.

Each modal operator is evaluated with respect to a pair of sets of interpretations. The idea is that $\mathbf{K}\varphi$ is true if φ is true in all elements in M; otherwise it is either false or undefined depending on M_1. If φ is true in all elements in M_1 then $\mathbf{K}\varphi$ is undefined; otherwise false. The case of $\mathbf{not}\varphi$ is handled symmetrically with respect to (N,N_1), the only difference is the condition for true for modal $\mathbf{K}$-atoms yields false for modal $\mathbf{not}$-atoms.

The included subset relation between M and N ensures the consistency of the interpretation pairs by not allowing a formula $\mathbf{K}\varphi$ to be true and false simultaneously within one interpretation, and, enforcing that whenever a formula φ is false, then $\mathbf{K}\varphi$ is false as well and not undefined.

Definition 4.10. Any interpretation pair (M, N) is a *partial (or 3-valued) MKNF model* for a given closed MKNF formula φ if:

1. $(I, < M, N >, < M, N >)(\varphi) = \mathbf{t}$ for all $I \in M$ and
2. $(I', \langle M', N' \rangle, \langle M, N \rangle)(\varphi) \neq \mathbf{t}$ for some $I' \in M'$ and each interpretation pair (M', N') with $M \subseteq M'$ and $N \subseteq N'$ where at least one of the inclusions is proper.

Definition 4.11. If there is a partial MKNF model of a given closed MKNF formula φ, then φ is called *MKNF-consistent* and *MKNF-inconsistent* otherwise.

In [35] it is shown that any (2-valued) MKNF Model M corresponds exactly to a (3-valued) partial one and vice-versa. In fact, the evaluation in MKNF structures (I, M, N) and total 3-valued structures $(I, < M, M >, < N, N >)$ is identical as nothing can be undefined in such a 3-valued structure.

Now, we present an order partial MKNF models ultimately claiming that the least element of this order is the well-founded model.

Definition 4.12. Let φ be a closed MKNF formula and (M_1, N_1) and (M_2, N_2) be a partial MKNF models of φ. Then $(M_1, N_1) \succeq_k (M_2, N_2)$ iff $M_1 \subseteq M_2$ and $N_1 \supseteq N_2$.

This order intuitively resembles the knowledge order where the least element contains the smallest amount of derivable knowledge, i.e. the one which leaves as much possible undefined.

4.2.4.1 Reasoning in 3-valued MKNF

The well-founded model for normal logic programs cannot be computed by an alternating fixpoint of the operator used to define stable models. In [34] it is proceed similarly: operators are defined in order to provide a stable condition for non-disjunctive hybrid MKNF knowledge bases, and used to obtain an alternating fixpoint from which it is possible to reconstruct the well-founded semantics.

Definition 4.13. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a hybrid MKNF knowledge base, the ground instantiation of $\mathcal{K}$ is the knowledge base $\mathcal{K}_G = (\mathcal{O}, \mathcal{P}_G)$ where $\mathcal{P}_G$ is obtained by replacing in each rule of $\mathcal{P}$ all variables with constants from $\mathcal{K}$ in all possible ways.

The set of $\mathbf{K}$-atoms of $\mathcal{K}$, written $\mathsf{KA}(\mathcal{K})$, is the smallest set that contains all $\mathbf{K}$-atoms occurring in $\mathcal{P}_G$, and a modal atom $\mathbf{K}\xi$ for each modal atom $\mathbf{not}\ \xi$ occurring in $\mathcal{P}_G$.

For a subset S of $\mathsf{KA}(\sigma)$, the objective knowledge of S is the formula $\mathsf{ob}_{\mathcal{K},S} = \mathcal{O} \cup \bigcup_{\mathbf{K}\xi \in S} \xi$, and $\widehat{S}_{DL} = \{\xi | \mathbf{K}\xi \in S_{DL}\}$ where S_{DL} is the subset of DL-atoms of S. A (partial) partition (T, F) of $\mathsf{KA}(\mathcal{K})$ is consistent if $\mathsf{ob}_{\mathcal{K},T} \not\models \xi$ for each $\mathbf{K}\xi \in F$.

The MKNF knowledge base must be modified in order to address the coherence problem. If a first-order formula φ is false (as a consequence of the DL part) then it must also be false by

default (i.e. $\mathbf{not}\varphi$ must hold in the program part). For this, instead of representing the connection directly, [34] introduces new positive DL atoms which represent the falsity of an already existing DL atom, and a further program, making these new atoms available for reasoning in the respective rules.

Definition 4.14. Let $\mathcal{K}$ be a DL-safe hybrid MKNF knowledge base. $\mathcal{K}^*$ is obtained from $\mathcal{K}$ by adding an axiom $\neg H \sqsubseteq NH$ for every DL atom $H(t_1, \ldots, t_n)$ which occurs as head in at least one rule in $\mathcal{K}$ where NH is a new predicate not already occurring in $\mathcal{K}$. Moreover, $\widehat{\mathcal{K}}$ is obtained from $\mathcal{K}^*$ by adding $\mathbf{not}NP(t_1, \ldots, t_n)$ to the body of each rule with a DL atom $P(t_1, \ldots, t_n)$ in the head.

The idea is to have $NH(t_1, \ldots, t_n)$ available as a predicate representing that $\neg H(t_1, \ldots, t_n)$ holds. $\mathcal{K}^*$ makes this connection explicit and $\widehat{\mathcal{K}}$ introduces a restriction on each rule with a DL atom in the head saying intuitively that the rule can only be used to conclude something if the negation of its head does not hold already.

It is now possible to define a monotonic operator $T_\mathcal{K}$ which allows reasoning from positive hybrid MKNF knowledge bases.

Definition 4.15. For $\mathcal{K}$ a positive non-disjunctive DL-safe hybrid MKNF knowledge base, $R_\mathcal{K}, D_\mathcal{K}$, and $T_\mathcal{K}$ are defined on the subsets of $\mathsf{KA}(\widehat{\mathcal{K}})$ as follows:

$$\begin{aligned} R_\mathcal{K}(S) = S \cup \{\mathbf{K}H \mid \mathcal{K} \text{ contains a rule of the form: } \mathbf{K}H \leftarrow \mathbf{K}A_1, \\ \ldots, \mathbf{K}A_n \text{ such that } \mathbf{K}A_i \in S \text{ for each } 1 \leq i \leq n\} \end{aligned} \tag{4.22}$$

$$\begin{aligned} D_\mathcal{K}(S) = \{\mathbf{K}\xi \mid \mathbf{K}\xi \in \mathsf{KA}(\widehat{K}) \text{ and } \mathcal{O} \cup \widehat{S}_{DL} \models \xi\} \cup \{\mathbf{K}Q(b_1, \ldots, b_n) | \\ \mathbf{K}Q(a_1, \ldots, a_n) \in S \setminus S_{DL}, \mathbf{K}Q(b_1, \ldots, b_n) \in \mathsf{KA}(\widehat{K}), \text{ and} \\ \mathcal{O} \cup \widehat{S}_{DL} \models a_i \approx b_i \text{ for } 1 \leq i \leq n\} \end{aligned} \tag{4.23}$$

$$T_\mathcal{K}(S) = R_\mathcal{K}(S) \cup D_\mathcal{K}(S) \tag{4.24}$$

The intuition is that $R_\mathcal{K}$ derives immediate consequences from the rules while $D_\mathcal{K}$ obtains consequences using modal atoms and the statements contained in the DL part. Note that $D_\mathcal{K}$ is parameterised over the inference model of S_{DL}. Since $T_\mathcal{K}$ is monotonic, it has a unique least fixpoint which is denoted $T_\mathcal{K} \uparrow \omega$, which is reached after a final number of iteration steps.

It is now necessary a transformation for non-disjunctive hybrid MKNF knowledge bases, turning them into positive ones and thus allowing the application of the operator $T_\mathcal{K}$.

Definition 4.16. Let $\mathcal{K}_G = (\mathcal{O}, \mathcal{P}_G)$ be a ground non-disjunctive DL-safe hybrid MKNF knowledge base and $S \subseteq \mathsf{KA}(\mathcal{K}_G)$. The MKNF transform $\mathcal{K}_G/S = (\mathcal{O}, \mathcal{P}_G/S)$ is obtained by $\mathcal{P}_G/S$ containing all rules $\mathbf{K}H \leftarrow \mathbf{K}A_1, \ldots, \mathbf{K}A_n$ for which there exists a rule:

$$\mathbf{K}H \leftarrow \mathbf{K}A_1, \ldots, A_n, \mathbf{not}B_1, \ldots, \mathbf{not}B_m$$

in $\mathcal{P}_G$ with $\mathbf{K}B_j \notin S$ for all $1 \leq j \leq m$.

This resembles the transformation known from stable models of logic programs and the following operator using a fixpoint of $T_{\mathcal{K}}$ is thus straightforward to understand.

Definition 4.17. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a non-disjunctive DL-safe hybrid MKNF knowledge base and $S \subset \mathsf{KA}(\widehat{K})$:

$$\Gamma_{\mathcal{K}}(S) = T_{\mathcal{K}^*_G/S} \uparrow \omega$$

$\Gamma_{\mathcal{K}}$ alone is not sufficient as, even though considering all modal atoms from $\mathsf{KA}(\widehat{\mathcal{K}})$, the applied knowledge base $\mathcal{K}^*$ does not enforce $\mathbf{not}H(t_1, \ldots, t_n)$ to hold if $\neg H(t_1, \ldots, t_n)$ holds. So it is needed a similar operator that refers to $\widehat{\mathcal{K}}_G$.

Definition 4.18. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a non-disjunctive DL-safe hybrid MKNF knowledge base and $S \subseteq \mathsf{KA}(\widehat{\mathcal{K}})$:

$$\Gamma'_{\mathcal{K}}(S) = T_{\widehat{\mathcal{K}}_G/S} \uparrow \omega$$

Both Γ and Γ' are shown to be antitonic [37] and form the basis for defining the well-founded MKNF model. Here we present its alternating fixpoint computation.

$$\begin{aligned} \mathbf{P}_0 &= \emptyset & \mathbf{N}_0 &= \mathsf{KA}(\mathcal{K}^*) \\ \mathbf{P}_{n+1} &= \Gamma_{\mathcal{K}}(\mathbf{N}_n) & \mathbf{N}_{n+1} &= \Gamma'_{\mathcal{K}}(\mathbf{P}_n) \\ \mathbf{P}_{\omega} &= \bigcup \mathbf{P}_n & \mathbf{N}_{\omega} &= \bigcap \mathbf{N}_n \end{aligned}$$

By finiteness of the ground knowledge base, the iteration stops before reaching ω. It was shown in [37] that the sequences are monotonically increasing, decreasing respectively, and that $\mathbf{P}_{\omega}$ and $\mathbf{N}_{\omega}$, can be used to reconstruct the well-founded model. Furthermore, $\mathbf{P}_{\omega}$ and $\mathbf{N}_{\omega}$ can also be classified as models T and TU, since $\mathbf{P}_{\omega}$ represent the model of all atoms that are *True*, whilst $\mathbf{N}_{\omega}$ represent the model of all the atoms that are *True and Undefined*. The remaining atoms that do not belong to $\mathbf{N}_{\omega}$ are false.

Theorem 4.1. *Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a DL-safe hybrid MKNF knowledge base and let $\mathbf{P}_{\mathcal{K}}, \mathbf{N} \subseteq \mathsf{KA}(\mathcal{K})$ with $\mathbf{P}_{\mathcal{K}}$ being $\mathbf{P}_{\omega}$ and $\mathbf{N}_{\mathcal{K}}$ being $\mathbf{N}_{\omega}$, both restricted to the modal atoms only occurring in $\mathsf{KA}(\mathcal{K})$. Let $(\mathbf{P}_{\mathcal{K}}, \mathbf{N}_{\mathcal{K}})$ be defined as the well-founded partition.*

Then the well-founded model of $\mathcal{K}$ is the pair $(I_{\mathbf{P}}, I_{\mathbf{N}})$ where:

$$\begin{cases} I_P = \{I : I \models \mathcal{O} \cup \bigcup_{\mathbf{K}\xi \in \mathbf{P}_{\mathcal{K}}}\} \\ \\ I_N = \{I : I \models \mathcal{O} \cup \bigcup_{\mathbf{K}\xi \in \mathbf{N}_{\mathcal{K}}}\} \end{cases}$$

All modal $\mathbf{K}$-atoms in I_P are true, all modal $\mathbf{not}$-atoms are false and all other modal atoms from I_N are undefined.

Moreover, for an empty DL base, the well-founded partition corresponds to the well-founded model for logic programs.

The data complexity result of computing the well-founded partition is $\mathsf{P}^{\mathcal{C}}$, where $\mathcal{C}$ is the data complexity of the DL used. This means that if the description logic fragment is tractable, this formalism can be computed with a data complexity of P. This is in contrast with the data complexity for reasoning with (two-valued) MKNF models in non-disjunctive programs, which is shown to be $\mathcal{E}^{\mathsf{P}^{\mathcal{C}}}$, where $\mathcal{E} = \mathsf{NP}$ if $\mathcal{C} \subseteq \mathsf{NP}$, and $\mathcal{E} = \mathcal{C}$ otherwise. Consequently, computing the well-founded partition generally ends up in a strictly smaller complexity class than deriving one of maybe various MKNF models. However, M is a (two-valued) MKNF model of $\mathcal{K}$ iff (M, M) is a 3-valued MKNF model of $\mathcal{K}$ and, furthermore, if (M, M) is the well-founded MKNF model of $\mathcal{K}$, then M is the only MKNF model of $\mathcal{K}$.

We conclude this section with some motivation examples for the 3-valued MKNF Semantics, showing how reasoning is performed in such cases.

Example 4.3. *Consider the following insurance example taken from [35]:*

$$NaturalDeath \sqsubseteq Pay \qquad Suicide \sqsubseteq \neg Pay$$

$$\begin{aligned}
\mathbf{K}\ Pay(x) &\leftarrow \mathbf{K}\ murdered(x), \mathbf{K}\ benefits(y,x), \mathbf{not}\ responsible(y,x)\\
\mathbf{K}\ Suicide(x) &\leftarrow \mathbf{not}\ NaturalDeath(x), \mathbf{not}\ murdered(x)\\
\mathbf{K}\ murdered(x) &\leftarrow \mathbf{not} NaturalDeath(x), \mathbf{not}\ Suicide(x)
\end{aligned}$$

This example relates to how a life insurance company decides to pay or not the insurance. Additionally, we know that Mr. Bill, who owned a life insurance, was found dead in his living room with the revolver still in his hand. Thus $\neg NaturalDeath(bill)$ *and the last two rules offer us a choice between commitment of suicide or murder. The standard MKNF semantics will immediately obtain two models in such scenario – one where* $pay(bill)$ *is true and other where* $pay(bill)$ *is false. However, the three-valued framework assigns* $undefined$ *to both, so that we delay this decision until some evidence is evaluated. Until then, by the first rule, no payment is possible.*

Furthermore, the police, which is also investigating the death of Mr. Jones, reveals that the known criminal Max is responsible for the murder, though not being detectable. So we cannot conclude $Suicide(jones)$*, while* $responsible(max, jones)$ *and* $murdered(jones)$ *hold. Unfortunately, the person benefiting from the insurance is the nephew Thomas who left the country many years ago.*

So, given the following facts and ABox:

$$\begin{aligned}
&\neg NaturalDeath(jones)\\
&\neg NaturalDeath(bill)\\
&responsible(max, jones).\\
&benefits(thomas, jones).\\
&benefits(ann, bill).\\
&murdered(jones).
\end{aligned}$$

The correspondent well-founded model is

$$T = \{responsible(max, jones), benefits(thomas, jones), benefits(ann, bill), murdered(jones), \neg naturalDeath(bill), \neg naturalDeath(jones), pay(jones)\}$$
$$Tu = \{responsible(max, jones), benefits(thomas, jones), benefits(ann, bill), suicide(bill), murdered(jones), murdered(bill), \neg naturalDeath(bill), \neg naturalDeath(jones), pay(bill), pay(jones)\}$$

However, a private detective is hired and he finds out that, in fact, Thomas is Max. So we add $max \approx thomas$*, stating that the person who killed Mr. Jones is the same that will benefit from his death. This situation violates the first rule of the knowledge base, and thus* $pay(jones)$ *is no longer contained in the model. The correspondent well-founded model is:*

$$T = \{\neg naturalDeath(jones), \neg naturalDeath(bill), benefits(max, jones), responsible(thomas, jones), murdered(jones), benefits(ann, bill), benefits(thomas, jones), responsible(max, jones)\}$$
$$Tu = \{\neg pay(bill), responsible(max, jones), benefits(thomas, jones), benefits(ann, bill), suicide(bill), murdered(jones), murdered(bill), responsible(thomas, jones), benefits(max, jones), \neg naturalDeath(bill), \neg naturalDeath(jones), pay(bill)\}$$

Example 4.4. *Consider now the following customs house example:*

$$NordicCountry \sqsubseteq SafeCountry \qquad DangerCountry \sqsubseteq Suspect$$

$$\mathbf{K}\ suspiciousContent(x, country) \leftarrow \mathbf{K}\ hasShipment(x, country), \mathbf{not}\ normal(x, country)$$
$$\mathbf{K}\ suspiciousContent(x, country) \leftarrow \mathbf{K}\ hasShipment(x, country), \mathbf{K}\ suspect(country)$$
$$\mathbf{K}\ dangerCountry(x) \leftarrow \mathbf{not}\ safeCountry(X)$$

This example simulates, in a simplified way, how a customs house decides whether a specific content is suspicious or not. Roughly, a content is suspicious if it comes from a country which is suspect, or it contains something that is not normal for the country.

So, afterwards we can have the following facts and ABox:

$$hasShipment(coconuts, norway).$$
$$hasShipment(coffee, brasil).$$
$$hasShipment(petrol, colombia).$$
$$normal(petrol, norway).$$
$$normal(codfish, norway).$$
$$normal(coconuts, brasil).$$
$$normal(coffee, brasil).$$
$$normal(coffee, colombia).$$
$$normal(petrol, colombia).$$
$$NordicCountry(Norway).$$

The well-founded model obtained from the evaluation of this Hybrid MKNF knowledge base:

$T = \{suspect(colombia), suspect(brasil), suspiciousContent(coffee, brasil),$
$suspiciousContent(petrol, colombia), dangerCountry(brasil), dangerCountry(colombia),$
$hasShipment(coconuts, norway), hasShipment(coffee, brasil),$
$hasShipment(petrol, colombia), normal(petrol, norway), normal(codfish, norway),$
$normal(coconuts, brasil), normal(coffee, brasil), normal(coffee, colombia),$
$normal(petrol, colombia), safeCountry(norway), nordicCountry(norway),$
$suspiciousContent(coconuts, norway)\}$
$Tu = \{suspect(colombia), suspect(brasil), suspiciousContent(coffee, brasil),$
$suspiciousContent(petrol, colombia), dangerCountry(brasil), dangerCountry(colombia),$
$hasShipment(coconuts, norway), hasShipment(coffee, brasil), hasShipment(petrol, colombia),$
$normal(petrol, norway), normal(codfish, norway), normal(coconuts, brasil),$
$normal(coffee, brasil), normal(coffee, colombia), normal(petrol, colombia),$
$safeCountry(norway), nordicCountry(norway), suspiciousContent(coconuts, norway)\}$

In this example, even though Norway is considered a safe country, it has a content that is not normal and consequently it is considered suspicious. On the other hand, since Brazil and Colombia are not defined as safe countries, they are considered as suspects and all their shipment is considered suspicious.

5 . Implementation of derivation methods

Previously, we presented a wide overview on the most important approaches for combining rules and ontologies, ending with the 3-valued MKNF Semantics. This semantics possesses some desirable decidability and complexity properties, but still preserving a considerable expressive power. In fact, the 3-valued MKNF Semantics is the first semantics able to address all the motivations that we pointed in Chapter 1, and maintain the complexity correspondent to the Description Logic employed.

So, now we continue to address the remaining proposed contributions by presenting a bottom-up and a goal-driven implementation of the 3-valued MKNF Semantics. To the best of our knowledge, these implementations represent the first implementations for knowledge bases that tightly combine nonmonotonic rules and DL ontologies, where the predicates of the rules can refer to propositions of the ontology, and these propositions may also refer to predicates defined in the rules. To perform such implementations, and since the 3-valued MKNF Semantics is parametric on any given DL, it was necessary to adopt a specific rule engine and an ontology manager, which led us to embrace XSB and CDF. XSB represents the natural choice for these implementations, first, because of its ability to compute queries according to the Well-Founded Semantics (through SLG resolution [54]), and second because it already comprehends an ontology management system – CDF [57]. On the other hand, CDF (Coherent Description Framework) is implemented over XSB. This framework is complete for the $\mathcal{ALCQ}$ DL and offers two types of instances each one with its associated complexity. Type-0 instances without negation and disjunctive, represent an ontology for which reasoning can be done with polynomial time algorithms; and Type-1 instances with full support for $\mathcal{ALCQ}$ – a DL for which reasoning is performed in `EXPTime`-complete, that extends $\mathcal{ALC}$ with qualified number restrictions. Yet, the main advantage in choosing XSB and CDF, is that, instead of picking two completely distinct systems that would suffer from the obvious integration problems, we adopt an extensive system that already includes the frameworks to deal simultaneously with rules and ontologies. Furthermore, these characteristics make it possible to perform the intended implementations given the particular time constraints of this thesis.

The definition of the 3-valued MKNF semantics is based on a fixpoint operator that, being monotonic, readily provides a bottom-up procedure for the calculus of the complete well-founded model. This way, we first present a bottom-up implementation for such semantics developed under XSB and CDF. This first solution is mainly based on the implementation of the several operators defined in the 3-valued MKNF semantics and represents the starting point of our work. In fact, this implementation permitted us to start playing with MKNF concepts, allowing the construction and experimentation of several examples. Moreover, this implementation serves as an experience and a comparison for our second and main solution – the goal-driven implementation for the MKNF well-founded semantics of hybrid knowledge bases.

The goal of the first implementation is to compute the *whole* model of the hybrid knowledge base, step by step. This approach may in fact be an advantage for small, uniform and strongly

interconnected knowledge bases. However, given the distributed infrastructure of the Semantic Web, which at a global scale is mainly dynamic and heterogeneous, in most situations it is naïve and unfeasible to compute the complete model of the knowledge base. Therefore, the need to define query-answering procedures arises, in order to support the extraction of knowledge in such context. These procedures intend to minimise the excess of computation by reducing the calculus of the model to what is strictly necessary, forming our second proposed solution.

The rest of this chapter shall proceed as follows: we start by presenting an overview of the platforms used for our implementations – XSB and CDF – referencing Tabling mechanisms and SLG procedures. Afterwards, we move forward to the two main contributions of this dissertation, a first bottom-up implementation and a goal-driven implementation.

5.1 Rule Engine – XSB

XSB [26] is an advanced logic based system, which extends the Prolog language by combining intelligent database technology with optimised logic programming technology. Among these features we highlight the SLG resolution and the handling of HiLog terms, that transform XSB into a new paradigm for logic programming.

In a typical Prolog system, SLD Resolution represents the common logic programming mechanism for the backtracking search through the tree of SLD refutations. However, the SLD computational mechanism, is clearly inadequate and unsatisfying in several situations, as it does not guarantee the termination.

In order to overcome the limitations of SLD resolution, rewriting techniques like Magic Sets [11] have been developed to take advantage of the bottom-up scheme. Here, rules are rewritten so that they may be implemented bottom-up (or forward chaining), in a way that cuts down on the irrelevant facts that are generated.

XSB offers a different approach [51], as, rather than depending on rewriting techniques, it extends SLD resolution in two ways: 1) adding tabling to make evaluations finite and non-redundant, and 2) adding a scheduling strategy and delay mechanisms to treat general negation efficiently. The resulting strategy is polynomial and is called SLG resolution [55]. SLG resolution, which is complete and finite for non-floundering programs with finite models, independently if they are stratified or not, empowers XSB with the ability to evaluate programs according to the Well-Founded Semantics (WFS).

5.1.1 Tabling

Tabling [61] (also called as memoization or lemmatization) in Prolog is a technique that can get rid of infinite loops for bounded-term-size programs and of possible redundant computations in the execution of Prolog programs, and therefore addressing the inadequacies of SLD.

At first, tabling may seem like a simple idea: programs are evaluated by storing newly found answers of current subgoals in a proper data space, called the table space. The method then uses

this table to verify for repeated calls to subgoals – whenever such a repeated goal is found, the subgoal's answers are recalled from the table instead of being re-evaluated against the program clauses.

From this simple description, one can easily see how tabling provides termination to various classes of Horn clause programs. However, tabling can be used to far greater effect than ensuring termination. One powerful feature of tabling is its ability to maintain other global elements of computation in the "table", such as information about whether one subgoal depends on another, and whether this dependency is through negation. By preserving this global information, tabling can be used to evaluate normal logic programs under the WFS. Here, the essential idea is that global information about dependencies is used to determine the truth value of literals that do not have derivation. If such literals are involved in a cyclic dependency through negation, they are undefined under WFS; if not, the literals belong to an unfounded set and are false in WFS. Furthermore, it can be shown that tabling allows non-floundering datalog programs with negation to terminate with polynomial data complexity under the WFS.

XSB was the first Prolog system to augment top-down depth-first computation with tabling, by using SLG Resolution [54] that we explain next.

5.1.2 SLG Resolution

SLG resolution is a tabling based method of resolution with polynomial time data complexity and that is sound and search space complete for all non-floundering queries under the well-founded semantics.

Summarily, SLG can be seen as a method that "partially evaluates" clauses relevant to a query by reducing them with respect to the well-founded model. Each answer in SLG is in fact a clause, whose body contains literals that would have to be true in order to derive the head. Because answers are clauses, the table produced by query evaluation can be seen as a *residual program*. After a query is marked as *complete*, if this residual is empty, the answer is classified as *unconditional*, otherwise it is considered *conditional* and undefined.

To provide the reader the appropriate background on this subject, we follow next the reformulated SLG proposed in [54].

Definition 5.1. An SLG evaluation consists in a sequence of forests of SLG trees. Nodes of SLG trees have the form:

Answer_Template :- Delay_set | *Goal_List*

or *fail*. In the first form, the *Answer_Template* is an atom used to represent bindings made to a tabled subgoal in the course of resolution along a computation path; the *Delay_List* contains a set of literals that have been selected by a fixed-order literal selection strategy but whose evaluation has been delayed; and the *Goal_List* is a sequence of unresolved literals. The second form is called a *failure node*. A node N is called an *answer* when it is a leaf node for which *Goal_List* is empty. If the *Delay_Set* of an answer is empty it is termed an *unconditional answer*, otherwise, it is a *conditional answer*.

Definition 5.2. (Delay Literals). A negative delay literal in the *Delay_Set* of a node N has the form $not\ D$, where D is a ground atom. Positive delay literals have the form D^{Call}_{Answer}, where A is an atom whose truth value depends on the truth value of some answer Answer for the subgoal Call. If θ is a substitution then $(D^{Call}_{Answer})\theta = (D\theta)^{Call}_{Answer}$.

Positive delay literals contain information so that they may be simplified when a particular answer to a given call becomes unconditionally true or false. It is useful to define answer resolution so that it takes into account the form of delay literals.

Definition 5.3. (Answer Resolution). Let N be a node A :- D | $L_1, \ldots, L_n$, where $n > 0$. Let *Ans* = A' :- D'| be an answer whose variables have been standardised apart from N, N is SLG resolvable with *Ans* if $\exists i, 1 \leq i \leq n$, such that L_i and A' are unifiable with an mgu θ. The SLG resolvent of N and *Ans* on L_i has the form:

$$(A \text{ :- } D \mid L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n)\theta$$

if D' is empty, and

$$(A \text{ :- } D, \overline{D} \mid L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n)\theta$$

where $\overline{D} = L_i$ if L_i is negative, and $\overline{D} = L_i{}^{L_i}_{A'}$ otherwise.

A set of subgoals is completely evaluated when it can produce no more answers. Formally,

Definition 5.4. A set $\mathcal{S}$ of subgoals in a forest $\mathcal{F}$ is completed evaluated if at least one of the conditions hold for each $S \in \mathcal{S}$

1. The tree for S contains an answer S :- | ; or
2. For each node N in the tree for S:
   (a) The selected literal L_S of N is completed or in $\mathcal{S}$; or
   (b) There are no applicable *NEW SUBGOAL, PROGRAM CLAUSE RESOLUTION, POSITIVE RETURN, DELAYING* or *NEGATIVE RETURN* operations for N.

Once a set of subgoals is determined to be completely evaluated, the *COMPLETION* operation marks the root node of the trees of each subgoal.

Definition 5.5. (Supported Answer). Let $\mathcal{F}$ be a SLG forest, S a subgoal in $\mathcal{F}$, and Answer be an atom that occurs in the head of some answer of S. Then Template is supported by S in $\mathcal{F}$ if and only if:

1. S is not completely evaluated; or
2. there exists an answer node *Answer :- Delay_Set* | of S such that for every positive delay literal D^{Call}_{Ans}, *Ans* is supported by *Call*.

An SLG evaluation consists of a (possibly transfinite) sequence of SLG forests. In order to define the behaviour of an evaluation at a limit ordinal, it is needed the notion of a least upper bound for a set of SLG trees. If a global ordering on literals is assumed, then the elements in the *Delay_Set* of a node can be uniformly ordered, and under this ordering a node of a tree can be taken as a term to which the usual definition of variance apply. In particular, nodes of SLG trees are treated as identical when they are variant.

A rooted tree can be viewed as a partially ordered set in which each node N is represented as $\{N, P\}$, in which P is a tuple representing the path from N to the root of the tree. When represented in this manner, it is easily seen that when T_1 and T_2 are rooted trees, $T_1 \subseteq T_2$ iff T_1 is a sub-tree of T_2. Furthermore, if T_1 and T_2 have the same root, their union can be defined as their set union, for T_1 and T_2 taken as sets.

Definition 5.6. Given a program P, an atomic query Q and a set of tabling operations, a tabled evaluation $\mathcal{E}$ is a sequence of SLG forests $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_\beta$ such that:

- $\mathcal{F}_0$ is the forest containing a single tree Q *:-* | Q
- For each successor ordinal, $n + 1 \leq \beta$, $\mathcal{F}_{n1}$ is obtained from $\mathcal{F}_n$ by an application of a tabling operation.
- For each limit ordinal $\alpha \leq \beta$, $\mathcal{F}_\alpha$ is defined as the set of trees T such that
  - The root of T, S *:-* |S is the root of some tree in a forest $\mathcal{F}_i$, $i < \alpha$; and
  - $T = \cup\{T_i | T_i \in \mathcal{F}_i, i < \alpha$ and T_i has root S *:-* |S $\}$

If no operation is applicable to $\mathcal{F}_\alpha$, $\mathcal{F}_\alpha$ is called a final forest of $\mathcal{E}$. If $\mathcal{F}_\beta$ contains a leaf node with a non-ground selected negative literal, it is floundered.

SLG forests are related to interpretations in the following manner.

Definition 5.7. Let $\mathcal{F}$ be a forest. Then the interpretation induced by $\mathcal{F}$, $I_\mathcal{F}$ has the following properties.

- A (ground) literal $A \in I_\mathcal{F}$ iff A is in the ground instantiation of some unconditional answer *Ans :-* | in $\mathcal{F}$.
- A (ground) atom $not\ A \in I_\mathcal{F}$ iff A is in the ground instantiation of a completely evaluated subgoal in $\mathcal{F}$, and A is not in the ground instantiation of any answer in $\mathcal{F}$

An atom S is successful in $\mathcal{F}$ if the tree for S has an unconditional answer S. S is *failed* in $\mathcal{F}$ if S is completely evaluated in $\mathcal{F}$ and the tree for S contains no answers. An atom S is successful (failed) in $I_\mathcal{F}$ if S' ($not\ S'$) is in $I_\mathcal{F}$ for every S' in the ground instantiation of S. A negative delay literal $not\ D$ is successful (failed) in a forest $\mathcal{F}$ forest if D is failed (successful) in $\mathcal{F}$. Similarly, a positive delay literal D^{Call}_{Ans} is successful (failed) in a $\mathcal{F}$ if *Call* has an unconditional answer *Ans :-* | in $\mathcal{F}$.

We will now define SLG operations as follows.

Definition 5.8. (SLG Operations). Given a forest $\mathcal{F}_n$ of a SLG evaluation of program P and query Q, where n is a non-limit ordinal, $\mathcal{F}_{n+1}$ may be produced by one of the following operations.

1. *NEW SUBGOAL:* Let $\mathcal{F}_n$ contain a non-root node

   $$N = \textit{Ans :- Delay\_Set} \mid G, \textit{Goal\_List}$$

   where G is the selected literal S or $not\ S$. Assume $\mathcal{F}_n$contains no tree with root subgoal S. Then add the tree S :- $\mid S$ to $\mathcal{F}_n$.

2. *PROGRAM CLAUSE RESOLUTION:* Let $\mathcal{F}_n$ contain a root node $N = S$:- $\mid S$ and C be a program clause *Head :- Body* such that *Head* unifies with S with mgu θ. Assume that in $\mathcal{F}_n$, N does not have a child $N_{child} = (S$:- $\mid$ *Body*$)\theta$. Then add N_{child} as a child of N.

3. *POSITIVE RETURN:* Let $\mathcal{F}_n$ contain a non-root node N whose selected literal S is positive. Let *Ans* be an answer node for S in $\mathcal{F}_n$ and N_{child} be the SLG resolvent of N and *Ans* on S. Assume that in $\mathcal{F}_n$, N does not have a child N_{child}. Then add N_{child} as a child of N.

4. *NEGATIVE RETURN:* Let $\mathcal{F}_n$ contain a leaf node

   $$N = \textit{Ans :- Delay\_Set} \mid not\ S, \textit{Goal\_List}.$$

   whose selected literal $not\ S$ is ground.

   (a) *NEGATION SUCCESS:* If S is failed in $\mathcal{F}$, then create a child for N of the form: *Ans :- Delay_Set* | *Goal_List*.

   (b) *NEGATION FAILURE:* If S succeeds in $\mathcal{F}$, then create a child for N of the form *fail*.

5. *DELAYING:* Let $\mathcal{F}_n$ contain a leaf node N = *Ans :- Delay_Set* | $not\ S$, *Goal_List*, such that S is ground in $\mathcal{F}_n$, but S is neither successful nor failed in $\mathcal{F}_n$. Then create a child for N of the form *Ans :- Delay_Set,* $not\ S$ | *Goal_List*.

6. *SIMPLIFICATION:* Let $\mathcal{F}_n$ contain a leaf node N = *Ans :- Delay_Set* | , and let $L \in$ *Delay_Set*

   (a) If L is failed in $\mathcal{F}$ then create a child *fail* for N.

   (b) If L is successful in $\mathcal{F}$, then create a child *Ans :- Delay_Set'* | for N, where *Delay_Set'* = *Delay_Set* $-L$.

7. *COMPLETION:* Given a completely evaluated set S of subgoals, mark the trees for all subgoals in S as completed.

8. *ANSWER COMPLETION:* Given a set of unsupported answers $\mathcal{UA}$, create a failure node as a child for each $Ans \in \mathcal{UA}$.

An interpretation induced by a forest has its counterpart for SLG. Using these concepts we can relate SLG to SLG evaluations.

Theorem 5.1. *Let $\mathcal{F}$ a forest in a terminated SLG evaluation of a query Q to a program P, and A an atom such that A :- A is the root of some tree in $\mathcal{F}$. Then*

$$WFM(P)|_A = \mathcal{I}_{\mathcal{F}}|_A$$

We refer the proof of this theorem to [56]. This theorem implies the correctness of SLG, and as such, proving correctness w.r.t. SLG will ensure the correctness w.r.t. the well founded model.

Example 5.1. *Consider the following example from [55]*

```
:- table t/0, p/0, q/0, r/0, s/0

t :- not p.
p :- q.
q :- not r.
r :- q, s.
s :- r.
```

As can be seen from Figure 5.1, the evaluation of the query t encounters a negative loop containing q and r. In order to determine the truth value of q and r in the well-founded model, the fixed left-to-right literal selection strategy must be broken so that other literals in the clause for r may be resolved. A *DELAYING* operation is applied to node 5, moving the selected literal $not\ r$ from the goal list to the delay list. Now, node 8 is a leaf that has no more unresolved literals in its goal list, so it is termed an answer (for q), but since its delay list is not empty, it is termed a *conditional answer*. This conditional answer is resolved against the selected literals of nodes 3 and 7. When a conditional answer A :- $DL|$ is resolved against a selected literal L, SLG does not propagate the elements of the delay list DL, but rather delays the selected literal, L, to indicate that a conditional answer was used for resolution. In non-propositional programs, this delayed literal may be subject to further unification operations so that positive delayed literals are annotated with the answer used for resolution, A, along with the root subgoal of the tree to which the answer belongs.

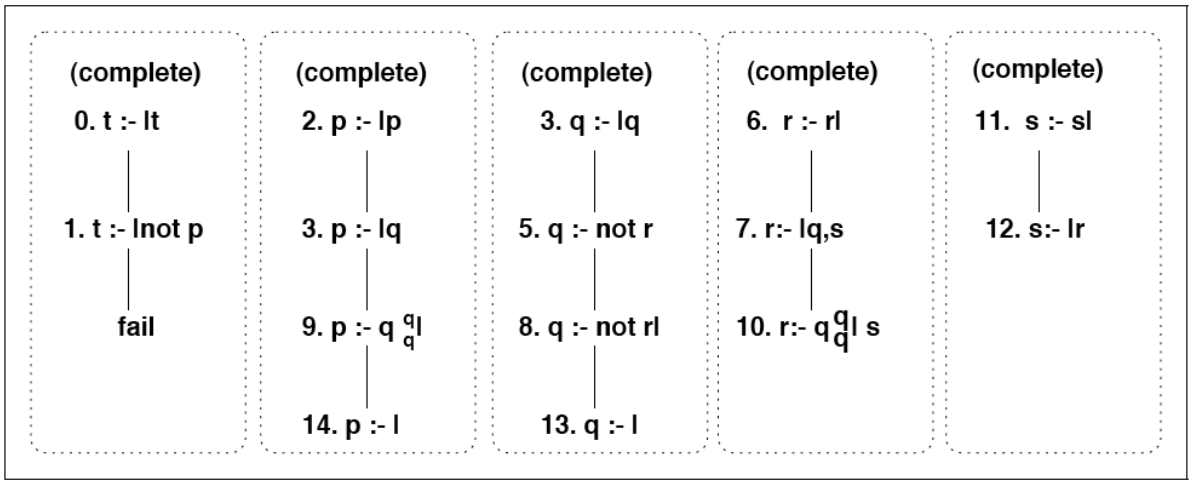

Figure 5.1 SLG Forest upon execution of the query ?- t.

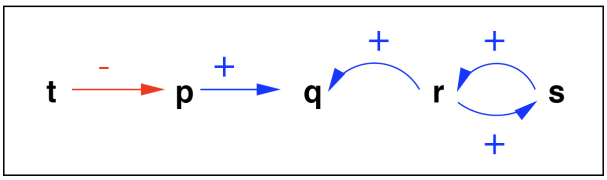

Figure 5.2 Subgoal Depedency Graph induced by execution of the query ?- t.

In this example, *ANSWER RETURN* adds the delayed literal q_q^q to the delay list of node 3 producing a conditional answer for p (node 9). Similarly, *ANSWER RETURN* of the conditional answer q *:-* $not\ r$ | is used for the selected literal of node 7 producing node 10. The evaluation then calls s, and recursively, r. Through a *COMPLETION* operation their trees are marked as *complete*. Upon the completion of r with no answers, since r is known to be false in the well-founded model, the answer q *:-* $not\ r$ | can be made unconditional. SLG *SIMPLIFICATION* operation is used to propagate truth values to delayed literals. Using *SIMPLIFICATION*, the delayed literal $not\ r$ is removed from the delay list of q's answer making this answer *unconditional* (node 13). The derivation of this unconditional answer enables a further *SIMPLIFICATION* operation to remove the delayed literal in the answer p *:-* q_q^q | for p. Finally, since p's truth value is now established, a *NEGATION RETURN* operation can be performed to fail the computational path leading to node 1 of the forest, and through *COMPLETION* operations, the remaining subgoals also become completed leading to the final forest represented in Figure 5.1.

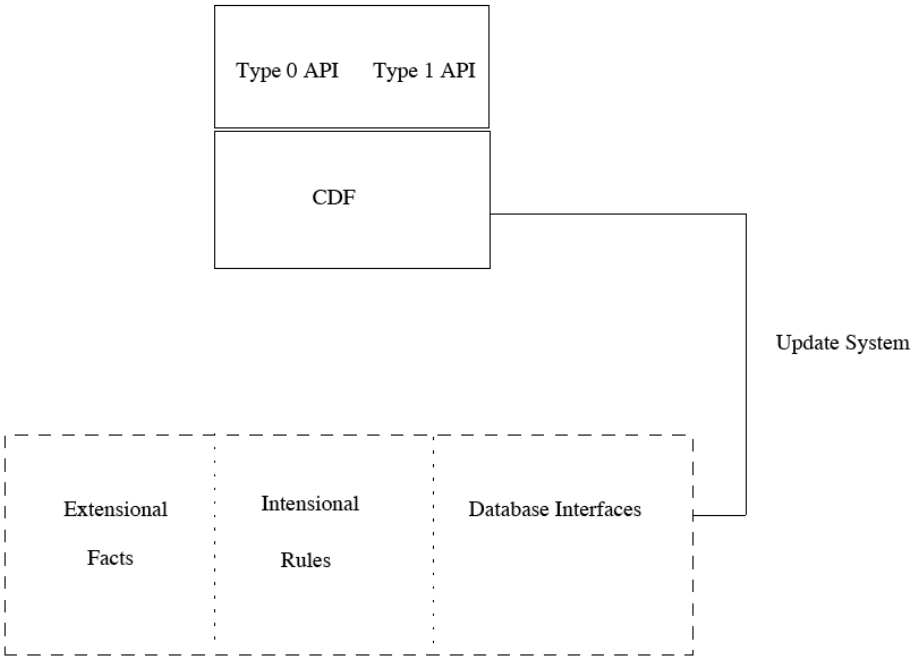

Figure 5.3 A High-Level Architecture of CDF

5.2 DL Framework – CDF

The Coherent Description Framework (CDF) [57] allows various sorts of support for management of formal ontologies from within XSB. This system is complete for the $\mathcal{ALCQ}$ DL, described before in Section 2.2.2, extended with relational hierarchies and product classes.

Since CDF is implemented over XSB, its architecture, represented in Figure 5.3, is already an advantage for a hybrid system – CDF stores knowledge in a *CDF Instance* consisting of information in the form of Prolog facts (called extensional facts), Prolog rules (called intensional rules), or in various database-resident formats. Furthermore, because of the tight integration of the two systems, CDF's implementation has also access to XSB's tables, that, as we shall see, will be particularly important in passing residual answers in the goal-driven implementation.

This way, there were two main reasons related to the choice of CDF. First, CDF represents a polynomial DL which empowers our solution of polynomial complexity (in the case of Type-0 instances), and second because, by its architecture, CDF has already means to define dependencies between rules and ontologies via *intensional rules*.

5.2.1 CDF Overview

CDF instances are divided in Type-0 and Type-1, each of which with its own interface. Type-0 instances are useful for storing large amounts of information as consistency and implication

in this kind of instances enjoy from polynomial complexity. These instances describe classes by existential and universal relations, qualified number restrictions, and relational hierarchies, allowing a direct product construction for objects and classes. However, they do not support negation and disjunction.

Type-1 instances extend Type-0 instances to describe classes using negation and disjunction and thus permit descriptions that are equivalent to an expressive description logic. Reasoning in this kind of expressive instances is done via the CDF theorem prover with a higher degree of complexity, now raised to `EXPTime`-complete.

One of the main advantages of CDF is the possibility to define intensional rules. A CDF instance is built up of extensional facts and intensional rules. Extensional facts allow to express ordinary description logic facts (ABoxes) and concepts (TBoxes and RBoxes). On the other hand, intensional rules are defined as XSB rules and, therefore, may use any XSB's language or library features, including tabling, database, and internet access. Intensional rules are called on demand, making them suitable for implementing functionalities from lazy database access routines to definitions of primitive types, as well as, to call prolog facts directly on the ontology.

5.2.1.1 CDF Syntax in a Nutshell

The basis of any description logic are Classes, Roles and Objects, which in CDF are defined, respectively, by class identifiers, denoted as $cid/2$, relation identifiers, $rid/2$, and object identifiers as $oid/2$.

The Type-0 CDF instances correspond to $\mathcal{ALCQ}$ DL without negation and disjunction, and for that, using the following constructors:

- $isa/2$ – defines relations between classes and classes and between classes and objects:

  ```
  isa(cid(class1,o1),cid(class2,o1)).
  isa(oid(object,o1),cid(class,o1)).
  ```

  Corresponding in DL notation to: $Class1 \sqsubseteq Class2$ and $object : Class$.

- $allAttr/3$ – indicates a typing for relations, but it does not require the existence of such a relationship. For instance, the next statement express that all objects that belong to the class *person* if they have a relation of the type *hasChild*, then this is to the class *person*.

  ```
  allAttr(cid(person,fam),rid(hasChild,fam),cid(person,fam).
  ```

  Which in DL syntax corresponds to: $Person \sqsubseteq \forall hasChild.Person$.

- $hasAttr/3$ – contrary to $allAttr/3$, this require that such relation exist between classes or objects. The next example shows that an object of a the class *person* has a relation *hasMother* to the class *mother*, and that the object *john* has a mother which is *mary*:

  ```
  hasAttr(cid(person,fam),rid(hasMother,fam),cid(mother,fam)).
  hasAttr(oid(john,fam),rid(hasMother,fam),oid(mary,fam)).
  ```

That is translated to $Person \sqsubseteq \exists hasChild.Person$ and $(john, mary) : hasMother$. This predicate provides a simple but powerful mechanism for inheritance of typing among CDF Classes.

- $classHasAttr/3$ – the $hasAttr/3$ relation described above implies that every subclass of *person* has a relation to the class *mother*. However, classes may have relations that do not hold for their subclasses or members. For example, a finite set may have a given cardinality, but its proper subsets will have a different cardinality. Such relations are termed as *class relations* and defined by $classHasAttr/3$

- $minAttr/4$, $maxAttr/4$ – These predicates define cardinality on roles. For instance, the next statement defines that a person has at least, and only one mother:

```
minAttr(cid(person,fam),rid(hasMother,fam),cid(mother,fam),1).
maxAttr(cid(person,fam),rid(hasMother,fam),cid(mother,fam),1).
```

  Which in DL is expressed by:

$$Person \sqsubseteq \leq 1 hasMother.Mother$$
$$Person \sqsubseteq \geq 1 hasMother.Mother$$

Type-1 CDF instances differ from Type-0 CDF instances as they provide full support to $\mathcal{ALCQ}$. This way, Type-1 instances allow a new kind of constructor: *necessCond/2*, that empowers Type-1 instances with disjunction and negation. For that, Type-1 instances have a new kind of identifier: *virtual identifier*, denoted by the functor $vid/1$. A virtual identifier indicates that rather than denoting a class by its name, it is denoted by a class expression whose syntax is given next.

Definition 5.9. Let $\mathcal{L}$ be an ontology language. A CDF *class expression* C over $\mathcal{L}$ is formed by one of the following constructions in which A is a class or object identifier, C_1 and C_2 class expressions, R a relation identifier, and n a natural number.

$$C \leftarrow A|not\ C_1|C_1, C_2|C_1; C_2|all(R, C_1)|exists(R, C_1)|atLeast(n, R, C_1)|atMost(n, R, C_1)$$

Corresponding, in DL notation to:

$$C \leftarrow A|\neg C_1|C_1 \sqcap C_2|C_1 \sqcup C_2|\forall R.C_1|\exists R.C_1|(\leq nR.C_1)|(\geq nR.C_1)$$

For instance, the following CDF instance:

```
necessCond(cid(s,ont),vid(';'(cid(p,ont),cid(q,ont))))
```

defines a class S that in DL syntax corresponds to $S \doteq P \sqcup Q$.

All the predicates, both in Type-0 and Type-1 CDF instances, can be extensional and intensional. Extensional instances are defined as facts and must be explicit in a *cdf_extensional.P* file and must be invoked with *load_extensional_facts(folder)*. Intensional instances are Prolog-like rules and must be defined in the file *cdf_intensional.P* and called as *load_intensional_rules(folder)*.

```
isa_ext(oid(john,fam),rid(hasFather,fam),oid(david,fam)).
hasAttr_int(oid(X,fam),rid(hasFriend,fam),oid(Y,fam)) :-
            hasAttr(oid(X,fam),rid(hasSibling,fam),oid(Y,fam)).
```

Here, the first statement represents a fact saying that *david* is the father of *john*, while the latter defines that siblings are friends. So, when we ask some query such as:

```
?- hasAttr(oid(X,fam),rid(hasSibling,fam),oid(Y,fam)).
```

The CDF system will check if X and Y can unify with some fact defined as $hassAttr\_ext/3$ fact, or if there is such rule in the intensional part such that:

```
hasAttr_int(oid(X,fam),rid(hasSibling,fam),oid(Y,fam))
```

5.2.1.2 The CDF Theorem Prover

In the current CDF version, a tableaux-style prover is used for entailment and to check consistency of class expressions. At a high level, the CDF prover first translates a class expression CE to a formula ψ in an ontology language. It then attempts to construct a model for ψ: if it succeeds, CE is consistent; otherwise CE is inconsistent (since the prover can be shown to be complete). The CDF prover has access to the relational and class hierarchies of a CDF instance during its execution. As a result, only the main classes and relations of an identifier need to be entered in class expressions.

Consistency in CDF can be checked using the follow system predicates:

- `checkIdConsistency/1` – In `checkIdConsistency(IdList)`, `IdList` is a (list of) class or object identifier(s) which is taken as a conjunction. The Predicate succeeds if `IdList` is consistent in the current CDF instance.

- `consistentWith/2` – In `consistentWith(Id,CE)`, `Id` can either be a class or an object identifier and `CE` is a class expression. This predicate checks whether `CE` is logically consistent with all that is known about `Id` in the current CDF instance. This determines whether there is a model of the current CDF instance that satisfies the expression `Id`, `CE`.

- `allModelsEntails/2` – In `allModelsEntails(Id,CE)`, `Id` is a class or object identifier and `CE` is a class expression. `allModelsEntails/2` succeeds if `CE` is entailed by what is known about `Id` in the current CDF instance. In other words, `allModelsEntails/2` determines whether in all models of the current CDF instance,

if an element is in `Id` then it is also in `CE`. It does this by checking the inconsistency of `Id`, `CE`.

Moreover, entailment can be performed by querying the ontology directly via the predicates `isa/2` and `hasAttr/3`. For instance, the query `isa(oid(Obj,N1),cid(Class,N2))` obtains all objects `Obj` that belong to a given class `Class`. This query does not only return what is explicitly defined as a class for a object, but also derives to what classes the object belongs to. So, if we have the following knowledge base:

```
isa_ext(oid(jones,fam),cid(professor,fam)).
isa_ext(cid(professor,fam),cid(person,fam)).
```

intuitively, Jones is a professor and every professor is a person. When checking entailment for Jones we obtain:

```
?- isa(oid(X,fam),cid(Y,fam)).

X = jones, Y = professor;

X = jones, Y = person;
no.
```

Analogously, we can use similar queries to retrieve entailment between classes, objects and relations. The query `isa(cid(Class1,N1),cid(Class2,N2))` returns all classes (`Class1`) that are a subset of some other class (`Class2`); and `isa(oid(Obj1,N1),oid(Obj2,N2))` returns all congruence relations between objects. Likewise, we can use `hasAttr/3` to check entailment between relations.

For a more detailed list of queries, we refer to Section 5.3, where we present all the queries performed by our solution in order to obtain entailment between classes, objects and relations.

5.3 Bottom-Up Implementation

Previously, in Section 4.2.4 we described the 3-Valued MKNF Semantics [34] that allows knowledge to be inter-definable between rules and any parametric ontology. As a contribution for this dissertation, we present herein a bottom-up implementation for such semantics.

This solution computes the well-founded model of a hybrid knowledge base with rules and ontology via an iterative fixpoint. The rules are defined as Prolog-rules that use the infix operator "`<-`":

```
H <- Body.
```

and with the ontology defined in the CDF syntax.

Given the interdependency of our knowledge base, as rules can have predicates that are defined on the ontology, and vice-versa, the main idea is, to infer knowledge simultaneously from both parts, iteratively, until a fixpoint is reached. For this goal, the 3-valued MKNF Semantics

in the Definition 4.15 defines three operators, $R_{\mathcal{K}}$, $D_{\mathcal{K}}$, $T_{\mathcal{K}}$, that are used in the calculus of the two iterative fixpoints Γ and Γ'. Similarly, this solution correspond to the implementation of these operators and these fixpoints, where the proof of correctness is in accordance with the proof for 3-valued MKNF Semantics in [35]. Moreover, given the complexity results from [35], it follows easily that our solution is empowered with polynomial complexity in the case of Type-0 ontologies.

To illustrate how our solution correspond to the definition in 4.2.4, recall the following operators defined previously in Page 47:

$$\begin{aligned}
R_{\mathcal{K}}(S) = S \cup \{\mathbf{K}H \mid \mathcal{K} \text{ contains a rule of the form: } \mathbf{K}H \leftarrow \mathbf{K}A_1, \\
\ldots, \mathbf{K}A_n \text{ such that } \mathbf{K}A_i \in S \text{ for each } 1 \leq i \leq n\} \\
D_{\mathcal{K}}(S) = \{\mathbf{K}\xi \mid \mathbf{K}\xi \in \mathsf{KA}(\widehat{K}) \text{ and } \mathcal{O} \cup \widehat{S}_{DL} \models \xi\} \cup \{\mathbf{K}Q(b_1, \ldots, b_n) | \\
\mathbf{K}Q(a_1, \ldots, a_n) \in S \setminus S_{DL}, \mathbf{K}Q(b_1, \ldots, b_n) \in \mathsf{KA}(\widehat{K}), \text{ and} \\
\mathcal{O} \cup \widehat{S}_{DL} \models a_i \approx b_i \text{ for } 1 \leq i \leq n\} \\
T_{\mathcal{K}}(S) = R_{\mathcal{K}}(S) \cup D_{\mathcal{K}}(S)
\end{aligned}$$

Identically, the core of our solution relies on these three operators (`r_op/5`, `d_op/3` , `t_op/6`) and two different sets – S and P – that represent the result of each fixpoint, where the first is the result of the inner fixpoint, and the latter the result of the outer of the double fixpoint of Γ and Γ' (cf. Page 48).

- `r_op/5` – given a S, P, a list of rules, and a flag, returns the Heads of the rules which can be proven. The Head H of a rule $H \leftarrow A_1, .., A_n, not\ B_1, not\ B_m$ is added to S if:

  1. $A_1, ..., A_n \in S$, $\neg\exists\ B_1, ..., B_m \in P$, flag = false or
  2. $A_1, ..., A_n \in S$, $\neg\exists\ B_1, ..., B_m \in P$, $\neg H \notin P$, flag = true.

  In this operator, the $flag$ is used to assure the difference between Γ and Γ'. Instead of implementing two different operators that are roughly the same, we opted by stating this difference via a parameter $flag$. In fact, Γ' compared to Γ only adds an extra-restriction, by assuring that if a proposition H is defined as explicitly false in the ontology, then $not\ H$ must hold as well. This is done by forcing all the rules with the head H to be ignored in case $\neg H$ holds. Consequently, when $flag = true$, an extra condition is checked – we only add H to our model if the previous conditions hold and if $\neg H$ does not belong to our set P.

- `d_op/3` – given S and a list of rules, returns the new information that can be derived from the ontology. Since S represents the current knowledge, this operator starts by asserting S in the ontology as an ABox:

  1. Unary terms – $p(t)$ are translated as an object t that belongs to a class p: $isa(oid(t, ont), cid(p, ont))$.

2. Binary terms – $p(t_1, t_2)$ correspond to a relation p between the object t_1 to t_2: $hasAttr(oid(t_1, ont), rid(p, ont), oid(t_2, ont))$.

Note that, since in an ontology it is only possible to express unary and binary terms, all the remaining terms like: $p(t_1, t_2, \ldots, t_n)$, where $n > 2$, are ignored by the ontology.

CDF does not offers any method for computing the complete model of the ontology. So, in order to retrieve all the information from the ontology, we need to perform a set of queries, that are translated to our program in the following way:

1. Getting all objects that belong to a given class, which is obtained by querying: $isa(oid(Obj, ont), cid(Class, ont))$, translated as $Class(Obj)$.
2. Obtained all relations between objects: $hasAttr(oid(Obj1, ont), rid(Relation, ont), oid(Obj2, ont))$ as $Class(Obj1, Obj2)$.
3. Retrieving all the congruence relations between objects: $isa(oid(Obj1, ont), oid(Obj2, ont))$. When this congruence restriction is satisfied, it will trigger the predicate $assertEqObjects/2$, that will add a copy from all rules and facts of $Obj1$ for $Obj2$, and vice-versa.
4. Collecting all objects that explicit do not belong to a given class: $isa(oid(Obj, ont), vid(not(cid(Class, ont))))$ as $not\ Class(Obj)$.

- `t_op/6` – calls `d_op/3` and `r_op/5` at each iteration.

Classical negation from the ontology is passed through the program by adding the predicate **not** $P(t_1, \ldots, t_n)$ as a fact. This **not** as a fact can only be derived by the ontology and is used to assure that $P(t_1, \ldots, t_n)$ does not hold if $\neg P(t_1, \ldots, t_n)$ is derived in the DL. In fact, the difference between the two fixpoints Γ and Γ' is to specifically guarantee this condition. So in Γ' a head H of a given rule is only added if $\neg H \notin P$. As stated before, in our solution this condition is ensured by a $flag$ that can be either $true$ or $false$ in the operators defined previously.

Corresponding to Γ and Γ' we define `dofixpointS/3` that given P and a $flag$, returns the set S. For that, it calls `t_op/6` to derive new knowledge iteratively. So S starts as empty and grows at each step until the fixpoint is reached. Now $\Gamma_{\mathcal{K}}(\Gamma'_{\mathcal{K}}(S))$ is done via `dofixpointP/2` that given the result S of `dofixpointS/3` with the $flag = true$ passes this S as P again to `dofixpointS/3` with the $flag = false$ and does this until the fixpoint is achieved. This latter operator, `dofixpointP/2` acts as a well-founded model operator, returning as an argument two sets: T and TU. These two sets correspond to $\mathbf{P}_{\mathcal{K}}$ and $\mathbf{N}_{\mathcal{K}}$ respectively from Definition 4.18. The first set (T) represents everything that is true in the well-founded model; while the latter (TU) represents everything that is true or undefined. The remaining predicates that do not belong to TU are false in the model. The basic idea of the algorithm can be seen in Figure 5.4.

Input: The set of all rules: $Rules$
Output: The set of all atoms that are true – P, and the set of all atoms that are true and undefined – S.

```
1  P = [];
2  while ( P ≠ P' ) do
3  |   P' = P;
4  |   S = [];
5  |   S' = t_op(S, P, Rules, true);
6  |   while ( S ≠ S' ) do
7  |   |   S = S';
8  |   |   S' = t_op(S, P, Rules, true);
9  |   end
10 |   P = S;
11 |   S = [];
12 |   S' = t_op(S, P, Rules, false);
13 |   while ( S ≠ S' ) do
14 |   |   S = S';
15 |   |   S' = t_op(S, P, Rules, false);
16 |   end
17 |   P = S;
18 end
```

Figure 5.4 General Bottom-Up Algorithm

5.3.1 Treatment of non-ground rules

The definition presented in Section 4.2.4 is only concerned with ground rules. However, when this definition is extended to rules that are non-ground, it is necessary to face the *floundering problem* related to the treatment of non-ground rules with negation in Prolog.

To give the reader a better understanding of this problem, let us consider a really simple Prolog program:

```
s(a).
s(b).
q(a).

p(X) :- not q(X), s(X).
```

The last rule states that `p(X)` is true if `q(X)` cannot be proven true on the program and if `s(X)` is true. If we inquire the program with the following queries, we obtain the answer presented:

```
?- p(a).
no

?- p(b).
yes

?- p(X).
no
```

The first two answers are as expected – `p(a)` is false because `q(a)` is true; and `p(b)` is true since `q(b)` cannot be derived in our program. However, the general query `p(X)` fails, when we would expect it to generate the answer `p(b)`. The reason for this problem relies on the implementation of the negation in Prolog which is known to be correct only for ground literals [40].

In fact, assuming closed world assumption, the negative goal $not\ q(X)$ in Prolog is interpreter as $not\ \exists x : q(x)$. That is, a request to find a term t such that $q(t)$ fails from the program. Thus, we would like Prolog to return bindings for the variable X which allow the call $q(X)$ to succeed. Unfortunately, the negation in Prolog is not able to generate such bindings for variables, but only to test whether a subgoal succeeds or fails. Consequently, in our example, X is unified with `a`, producing the answer `no`, and losing the remaining answers. To avoid this behaviour, the negation operator must only be applied to ground literals and when it does not, the program is said to *flounder*.

Identically, this problem arises when computing the bottom-up model. Consider, for instance the following program:

```
a(X) <- not b(X), p(X).

b(jones) <- true.
```

```
p(jones) <- true.
p(bill)  <- true.
```

When computing this program in a bottom-up fashion way, as described in the previous section, we obtain the following model:

$$T = [p(bill), p(jones), b(jones)]$$
$$Tu = [p(bill), p(jones), b(jones)]$$

However, $a(bill)$ should also be contained on the model as its body – $p(bill)$ and $not\ b(bill)$ succeed as well.

To understand the problem, recall that $Head$ is added to the model if the rule $Head \leftarrow A_1, \ldots A_n, \neg B_1, \ldots, \neg B_n$ exists and if $B_1, \ldots B_n \notin P$. Nevertheless, for the rule $a(X)$, we have to prove that $b(X)$ does *not* belong to P. Since this negation can only be applicable to ground predicates, $jones$ is instantiated with X and $b(jones) \in P$, making the rule to fail. Because we are testing a negative condition, Prolog cannot generate further bindings for X and the remaining answers are lost.

5.3.1.1 Solution

Solving the floundering problem is far from being a trivial task, and therefore, it is not in the scope of our work. However, this can be ignored if the programmer provides bindings to the variables *before* applying the negation-as-failure operator. For instance, consider the following program:

```
s(a).
s(b).
q(a).

p(X) :- s(X), not q(X).
```

Although this program looks equivalent, now the previous problem does no longer apply. The reason is that `s(X)` occurs before the negation, generating bindings to X, which will be ground in `q(X)`.

Furthermore, we recall that a 3-valued MKNF Hybrid Knowledge Base must respect the concept of *DL-safe*, described in Section 4.2.1. DL-safeness limits the application of the rules to known individuals. In practice, this means that the predicates $O(x_1), O(x_2), \ldots, O(x_n)$, which are only defined in the Rules-component, are added to each rule, for each variable x_n occurring in the given rule.

Consequently, we can assume that a positive predicate occurs in each rule for each variable, and therefore, if we impose these positive predicates to appear before the negative ones, then the positives will generate bindings for each variable and the negation is only applied to ground literals forcing the program to not *flounder*. This way, to avoid the problem, before applying any operation, our solution performs this *ordering* by a pre-processment of the rules.

5.3.2 Usage

A Hybrid Knowledge base is defined over a XSB-Prolog knowledge base together with an ontology specified over CDF. This is translated to the need of defining three different files for the hybrid knowledge base to be loaded as a whole:

- $rules.P$ – containing the set of Prolog rules and Prolog facts. A rule is defined with the operator "<–" as:

  ```
  Head <- Body.
  ```

  and

  ```
  Head <- true.
  ```

  representing a fact. As stated before, each rule must respect DL-safeness, in order to the Hybrid Knowledge Base to be DL-safe.

- $cdf\_extensional.P$ – comprising ordinary ontology facts and concepts defined over the CDF syntax described in 5.2.1.1.

- $cdf\_intensional.P$ – including *intensional rules* that are defined as Prolog-like rules:

  ```
  Head :- Body.
  ```

  but where the $Head$ represents a particular ontology fact or concept that holds when the $Body$ holds as well. The $Body$ is a set that can refer to any library feature as well as any CDF predicate.

An atom A can be defined as a predicate, as a proposition, or both. Moreover any predicate can be defined over a proposition as well as any proposition can be defined over a predicate. So, all atoms are freely stated either in $rules.P$ or/and in $cdf\_extensional.P$/$cdf\_intensional.P$, and thus, it is the responsibility of our algorithm to check where A is defined in order to retrieve the correct well-founded MKNF model.

5.3.3 Examples

Next we will show how the examples, presented before in Section 4.2.4, are mapped into XSB and CDF syntax, in order to be evaluated by our bottom-up algorithm.

Example 5.2. *Recall the following insurance example – Example 4.3 from [35]:*

$$NaturalDeath \sqsubseteq Pay \qquad Suicide \sqsubseteq \neg Pay$$
$$\mathbf{K}\ Pay(x) \leftarrow \mathbf{K}\ murdered(x), \mathbf{K}\ benefits(y,x), \mathbf{not}\ responsible(y,x)$$
$$\mathbf{K}\ Suicide(x) \leftarrow \mathbf{not}\ NaturalDeath(x), \mathbf{not}\ murdered(x)$$
$$\mathbf{K}\ murdered(x) \leftarrow \mathbf{not} NaturalDeath(x), \mathbf{not}\ Suicide(x)$$

This MKNF knowledge base is then translated to CDF syntax:

```
isa_ext(cid(naturalDeath,ont),cid(pay,ont)).
isa_ext(cid(suicide,ont),vid(not(cid(pay,ont))).
```

And to the following set of rules:

```
pay(X)      <- murdered(X), benefits(Y,X), not responsible(Y,X), p(X),
    p(Y).
suicide(X) <- not naturalDeath(X), not murdered(X), p(X).
murdered(X)<- not naturalDeath(X), not suicide(X),  p(X).
```

Note that the predicate $p(X)$ is an auxiliary predicate that is only defined in the rules. This way, this predicate was introduced in order to assure that the rules are only applied to known individuals, and thus assuring the DL-safety in our program.

Now, inserting $\neg NaturalDeath(jones), \neg NaturalDeath(bill)$ as an ABox:

```
isa_ext(oid(bill,ont),vid(not(cid(naturalDeath,ont)))).
isa_ext(oid(jones,ont),vid(not(cid(naturalDeath,ont)))).
```

And the following facts:

```
responsible(max,jones) <- true.
benefits(thomas,jones) <- true.
benefits(ann,bill)     <- true.
murdered(jones)        <- true.

p(jones)  <- true.
p(thomas) <- true.
p(bill)   <- true.
p(ann)    <- true.
```

```
isa_ext(oid(max,ont),oid(thomas,ont)).
```

Example 5.3. *Recall now the customs house example – Example 4.4:*

$$NordicCountry \sqsubseteq SafeCountry$$
$$DangerCountry \sqsubseteq Suspect$$

With the following set of rules:

$$\mathbf{K}\ suspiciousContent(x, country) \leftarrow \mathbf{K}\ hasShipment(x, country), \mathbf{not}\ normal(x, country)$$
$$\mathbf{K}\ suspiciousContent(x, country) \leftarrow \mathbf{K}\ hasShipment(x, country), \mathbf{K}\ suspect(country)$$
$$\mathbf{K}\ dangerCountry(x) \leftarrow \mathbf{not}\ safeCountry(X)$$

Which is translated for the following TBox in the CDF syntax:

```
isa_ext(cid(nordicCountry,ont),cid(safeCountry,ont)).
isa_ext(cid(dangerCountry,ont),cid(suspect,ont)).
```

And to the following set of DL-Safe rules:

```
suspiciousContent(X,Country) <- hasShipment(X,Country),not normal(X,
   Country), c(Country), s(X).
suspiciousContent(X,Country) <- hasShipment(X,Country), suspect(
   Country), c(Country), s(X).
dangerCountry(X) <- not safeCountry(X), c(X).
```

If we now add $NordicCountry(Norway)$ *to the ABox:*

```
isa_ext(oid(norway,ont),cid(nordicCountry,ont)).
```

And the set of facts:

```
hasShipment(coconuts,norway) <- true.
hasShipment(coffee,brasil)   <- true.
hasShipment(petrol,colombia) <- true.
normal(petrol,norway)        <- true.
normal(codfish,norway)       <- true.
normal(coconuts,brasil)      <- true.
normal(coffee,brasil)        <- true.
normal(coffee,colombia)      <- true.
normal(petrol,colombia)      <- true.

c(brasil)   <- true.
c(colombia) <- true.
c(norway)   <- true.
s(petrol)   <- true.
s(coffee)   <- true.
s(coconuts) <- true.
s(codfish)  <- true.
```

As expected, our bottom-up implementation evaluates both examples as described in Section 4.2.4. The source code of our implementation along with these and additional examples can be downloaded in `http://pessoa.fct.unl.pt/asg19136/bottomup.zip`.

5.4 Goal-driven Implementation

In the previous section we presented a bottom-up implementation for the 3-valued MKNF Semantics which computes the complete well-founded model for a given hybrid knowledge base, step by step, via an iterative monotonic fixpoint. However, given the sheer size and distributed nature of the Semantic Web, deriving all consequences of a knowledge base is mainly a naïve

and, in most cases, an impractical task. This way, considering the context of the Semantic Web, when the goal is to answer a specific query, there is a need to define lighter mechanisms relying on top-down/goal-driven procedures in order to minimise the computation to the set of literals of which the given query depend on.

In Section 5.1.2 we presented SLG resolution, employed by XSB, that already evaluates logic programs according to the well-founded semantics via a goal-driven procedure. However, now it is necessary to consider that a query can depend on both rules and ontology, where the dependency is such that a rule can use predicates that are defined in the ontology and the ontology can use propositions that are also defined in the rules.

Herein we present our main contribution, which is the first implementation of a hybrid knowledge base with such properties. Our solution makes use of XSB's SLG Resolution for the evaluation of a query, together with tableaux mechanisms supported by CDF theorem prover to check entailment on the ontology. It maintains the full compatibility corresponding to the 3-valued MKNF Semantics presented in Section 4.2.4. In particular, if our hybrid knowledge base does not contain any rules, we obtain the same as having our ontology defined in CDF, and if our ontology is empty, the result is the well-founded model for the given program.

5.4.1 Description

In our algorithm, a query is evaluated through SLG resolution until it depends on a given proposition defined in the ontology. Then the computational is passed to the ontology, which will use tableaux mechanisms to test this proposition, which may itself depend on some other rule literal. Nevertheless, this query-evaluation procedure cannot be done straightforwardly in order to assure completion/termination.

In its essence, a tableau algorithm decides the satisfiability of A w.r.t. a KB $\mathcal{O}$ by trying to construct a common *model* for A and $\mathcal{O}$, called a *completion graph*. If it succeeds, A is satisfiable, otherwise A is unsatisfiable. As stated before, in Section 5.2.1.2, the CDF theorem prover only constructs a model for the classes and relations of a specific identifier. So given a query Q, the CDF theorem prover constructs a model for all the properties of the object related to Q and checks if Q is contained in that model.

Now, given the particular integration between rules and the ontology, we have to consider the knowledge inferred in the rules for this satisfiability test, as a proposition can depend on some predicate of the rules, which however may rely on other ontology proposition. Thus, considering that a query relates to a particular object, the main idea of our solution is to compute, iteratively, a model for this object, deriving at each iteration new information about the given object either in the ontology (via CDF's tableau algorithm), or in the rules (via SLG procedures), until a fixpoint is reached.

We start by considering the special case of positive knowledge bases without default negation in the rules:

Example 5.4. *Consider the following DL-safe Hybrid MKNF Knowledge Base and the query*

$third(X)$:

$$\mathbf{K}\, third(X) \leftarrow \mathbf{K}\, p(X), \mathbf{K}\, second(X).$$
$$first(callback).$$
$$p(callback).$$
$$First \sqsubseteq Second$$

The evaluation of the query starts to find the rule for $third(X)$ which the body depends on the predicates $p(X)$ and $second(X)$. The predicate p, only defined in the rules, assures DL-safety, restricting the application of the rules to known individuals. Thus, the call $p(X)$ returns true for $X = callback$. However, now the call $third(callback)$ (since X was ground to $callback$ by p) depends on a proposition from the ontology – $second$. So the computation calls CDF theorem prover which starts to derive a model for all the properties of the object $callback$. Yet, in this computation, the proposition $second$ itself depends on a predicate defined in the rules – $first$. As a result, it is intuitive for such examples that the evaluation of the query $third(callback)$ must be done iteratively – the call $third(X)$ should hold (as in SLG procedures) until p(X) and $second(X)$ is resolved. Furthermore, $second(X)$ needs first to prove $first(callback)$ from the rules. Finally, since $second(callback)$ succeeds, the conditional answer $third(callback)$ becomes unconditional and succeeds for $X = callback$.

So, the idea of our solution is that the two components use each other as a way to derive new knowledge, interchanging what they know at each moment to the other component. This way, our algorithm goes by computing a model for the given object related to the query, iteratively, by inquiring both components. At the end of each iteration, the two components share the knowledge inferred. However, since it is possible to define n-ary predicates and roles, the query may not depend only on one object, but rather in a group of objects. Therefore, the model must be constructed taking into account a sets of object that the query depends on. This is done by maintaining a *table* with this set of objects that need to be computed in order to answer a given query. In a top-down fashion, this set increases as new dependency relations between objects are disclosed. The iteration stops when it is not possible to derive anything more about these objects, i.e., when all objects have reached the fixpoint.

Definition 5.10. Let $\mathcal{K} = (\mathcal{O}, \mathcal{P})$ be a DL-safe hybrid MKNF knowledge base. Consider $\mathcal{P}^+$ a subset of $\mathcal{P}$ containing only positive rules. Let $\mathcal{O}'$ and $\mathcal{P}'$ correspond to the subset of $\mathcal{O}$ and $\mathcal{P}^+$, respectively, necessary to compute in order to answer a query Q. Let R be the model obtained by the Rules-Component and D the model obtained by the DL-Component. The function $Tableaux$ relates to the application of tableaux algorithms to infer a model for the ontology. SLG relates to SLG procedures used by XSB to evaluate the queries correspondent

to P'. The model is obtained as follows:

$$\begin{array}{ll} D_0 = Tableaux(\mathcal{O}') & R_0 = SLG(\mathcal{P}') \\ D_1 = Tableaux(\mathcal{O}' \cup R_0) & R_1 = SLG(\mathcal{P}' \cup D_0) \\ \vdots & \vdots \\ D_n = Tableaux(\mathcal{O}' \cup R_{n-1}) & R_n = SLG(\mathcal{P}' \cup D_{n-1}) \end{array}$$

where n is odd and ≥ 2. The iteration stops when a fixpoint in R_n is reached.

This definition resembles the definition of the operator $T_{\mathcal{K}}$ presented in [34]. Similarly, since we are considering only positive rules, the operators SLG and $Tableaux$ are monotonic and thus, by the Knaster-Tarski theorem [59], the program yields a least fixpoint. Furthermore, the program respects DL-safeness, which means that MKNF rules are grounded with respect to the set of individuals (constants), and thus the program is finite and the fixpoint can be obtained in a finite number of steps.

Nevertheless, given the disparity of the essence of the components, as rules adhere to closed-world assumption, while the ontology assumes open-world assumption, this iteration is not enough when adding negation. As an illustration, consider the following example.

Example 5.5. *Consider the following DL-safe Hybrid MKNF Knowledge Base and the query* $third(X)$*:*

$$\begin{array}{l} \mathbf{K} \quad third(X) \leftarrow \mathbf{K}\, p(X), \mathbf{K}\, second(X). \\ \mathbf{K}\, fourth(X) \leftarrow \mathbf{K}\, p(X), \mathbf{dlnot}\, third(X). \\ first(callback). \\ p(callback). \\ First \sqsubseteq Second \qquad\qquad Fourth \sqsubseteq Fifth \end{array}$$

Now in Example 5.5, we have a predicate $fourth$ that is defined at the expense of the negation of $third$. Since this predicate is defined in the rules, the negation is *closed world*, that is, $fourth$ should only succeed if it is not possible to prove $third$ in our knowledge base. Consequently, if we employ SLG resolution blindly, we would consider $fourth$ as true, even though $third$ could be proven afterwards, when considering all the knowledge inferred by the ontology. Likewise, the rules may pass to the ontology knowledge, that after some iterations, no longer applies – $Fifth$ should only succeed in the case that $fourth$ is true, which is defined over closed world assumption.

From Example 5.5 it follows a need to treat nots carefully, as a not requires to be constantly re-evaluate by the inference of new knowledge. Recall the definition of the 3-valued MKNF semantics. In this semantics, an operator Γ is defined in order to address the problem of closed-world negation. Roughly, Γ is defined as the application of $T_{\mathcal{K}}$ until achieving a fixpoint. Applying Γ twice is a monotonic operation and thus yields as well a least fixpoint by the

Knaster-Tarski theorem. In each dual application of Γ two different models follow – a monotonically increasing model of trues (i.e. true predicates and propositions), and an monotonically decreasing model of trues and undefineds.

In a similar way, our solution requires the computation of two fixpoints. An *inner* fixpoint where we apply the Definition 5.10, and an *outer* fixpoint for the evaluation of nots. These inner and outer fixpoint correspond to the definition of $T_{\mathcal{K}}$ and $\Gamma_{\mathcal{K}}$, respectively. In $\Gamma_{\mathcal{K}}$, the evaluation of closed-world negation is made by a reference to the last model obtained by $\Gamma_{\mathcal{K}}$. Identically, in our solution, $not(A)$ succeeds if, in the last *outer* iteration, A was not proven.

Example 5.6. *As an illustration of the need of the application of the two fixpoints, consider the following DL-safe Hybrid MKNF Knowledge Base and the query $c(X)$:*

$$\mathbf{K}\ c(X) \leftarrow \mathbf{K}p(X), \mathbf{K}\ a(X), \mathbf{not}\ b(X)$$
$$p(object)$$
$$a(object).$$
$$A \sqsubseteq B$$

Considering the query $c(X)$, X is grounded to $object$ by the predicate p which, again, assures DL-safety. In a first inner iteration, $a(object)$, $p(object)$ and $c(object)$ succeed; simultaneously, the ontology is not able to prove anything since the proposition A does not hold. In the end of this iteration, the two components share knowledge as described in Definition 5.10. In a second inner iteration the rules maintain what they inferenced before and the ontology derives A. After sharing this knowledge, there is no more to infer by either components, and a fixpoint is reached. When the inner fixpoint is reached, we end the first outer iteration. So now, the second outer iteration will start the computation of the inner iteration again and, in this iteration, nots are evaluated given the last outer iteration. As a consequence, $c(object)$ fails, since $b(object)$ is proved in the latter iteration. The inner iteration fixes by inferring $p(object)$, $a(object)$ and $b(object)$, which is in fact the correct model for the object $object$. Afterwards, the outer iteration needs one more computational in order to stop, returning the model described. Since $c(object)$ is in the model, the query $c(X)$ returns true for $X = object$.

Summarily, in order to resolve a query, our solution uses SLG procedures in each inner iteration of the model to derive knowledge from the rules-component and tableaux mechanisms to infer knowledge from the ontology. To derive negation as failure of an atom A we use the predicate $dlnot(A)$, which succeeds if A was not proven in the last outer iteration. Contrary to the bottom-up approach, this computation is limited to the objects related to the query and performed via the predicates $known/2$, and $allModelsEntails/2$ as described in Figure 5.5. The first predicate infer knowledge from the rules-component using SLG procedures, whilst the latter derives knowledge from the DL-component via tableaux reasoning. The predicates $definedClass/2$ and $definedRole/3$ represent the domain of all classes and roles defined over a DL-safe MKNF Hybrid Knowledge Base. We assume that these predicates are defined

explicitly by the programmer, but they could also be inferred via DL-safe restriction. In fact, by bounding our program to DL-Safe rules, every rule in the hybrid knowledge base must contain a positive predicate that is only defined in the rules. This predicate limits the evaluation of the rules to *known* individuals. In practical, this means that we can infer the set of individuals that are applicable to each rule, that is, our *domain*.

Furthermore, since we can only express unary and binary relations in DL syntax, for the remaining n-ary predicates, SLG resolution is used as usual.

This process is described in the algorithm of Figure 5.5. In it, we obtain two different models related to the application of the operator Γ of the 3-valued MKNF semantics [34]. An optimistic model, containing the predicates and propositions that *might* be true; and a sceptical model comprising the predicates and propositions that *necessarily* hold. These models correspond to models of TU (trues and undefineds) and T, respectively that, given the architecture of our algorithm, are obtained in different outer iterations. As in the application of Γ, the T model is monotonically increasing, while TU is monotonically decreasing.

Finally, after computing the models and achieving the fixpoint, our algorithm returns the evaluation of $known(Query, Iteration - 1)$, where $Iteration$ represents the iteration where the outer fixpoint was accomplished. Since the first outer model obtained corresponds to the first iteration in the TU model, this outer fixpoint will be obtained in a TU iteration. Thus, in order to check if the query is true, we need to check if it contained in the model inferred in $Iteration - 1$. If this is not the case, the query is evaluated as undefined if it derived in $Iteration$, and false otherwise.

Next, we provide a detailed description of each component of the algorithm. We start by reviewing how knowledge is inferred by the Rules-component and the DL-component. Afterwards, we focus on how the interaction of the two components is performed, finishing with a description of how the fixpoint model is obtained.

5.4.1.1 Rules-Component

The knowledge is derived from the Rules-Component via the predicates $known/2$ and $dlnot/2$, where SLG resolution is employed as usual. Therefore, the evaluation of $Term$ in rules is roughly the execution of the call $Term$ in the program. Intuitively, the for the predicate $known(Term, MajIter, MinIter)$ to succeed, $Term$ has to succeed in the given $MajIter$ and $MinIter$. Yet, even if the call cannot be proven in the rules, it still can be true if the ontology derives $Term$, and hence, the predicate $known(Term, MajIter, MinIter)$ also succeeds if $lastAllModelsEntails(Term)$ is true, i.e., if $Term$ was derived in the last inner iteration of the ontology.

However, we still have to assure that if $\neg A$ holds, then $not\ A$ holds as well. In SLX [1], a top-down procedure for extended logic programs, this is guaranteed by forcing on TU-trees to prove $not\ \neg A$ when proving A. Identically, when we try to derive the predicate $known(Term, MajIter, MinIter)$ in a outer TU iteration, i.e., when the iteration $MajIter$

```
   Input: A query Query
   Output: True value of the input query
 1 addObjects(Query,Table);
 2 foreach Object in Table do
 3     MajIter, MinIter = 0;
 4     S = {};
 5     P = {};
 6     repeat
 7         P1 = P;
 8         repeat
 9             S1 = S  foreach Class in definedClass(Object,Class) do
10                 Term = Class(Object);
11                 S1 = S1∪ known(Term, MajIter, MinIter);
12                 S1 = S1∪ allModelsEntails(Term, MajIter, MinIter);
13                 S1 = S1∪ allModelsEntails(not Term, MajIter, MinIter);
14             end
15             foreach Role in definedRole(Object,Object1,Role) do
16                 Term = Role(Object,Object1);
17                 S1 = S1∪ known(Term, MajIter, MinIter);
18                 S1 = S1∪ allModelsEntails(Term, MajIter, MinIter);
19                 S1 = S1∪ allModelsEntails(not Term, MajIter, MinIter);
20             end
21             MinIter++;
22         until S = S1 ;
23         P = S;
24         MajIter++;
25     until P = P1 ;
26 end
27 if known (Query,Final-1,Final) then
28     return true
29 else
30     if known(Query,Final,Final) then
31         return undefined
32     else
33         return false
34     end
35 end
```

Figure 5.5 General Top-Down Algorithm

is even, we first check if the ontology derived $\neg Term$ in the last model. If so, then the call $known(Term, MajIter, MinIter)$ automatically fails. This restriction is imposed by the predicate $prev\_neg/2$.

The predicate $known/2$ correspond to the following code:

```
:- table known/2.
known(Term,MajIter,MinIter):-
  ( (MinIter = 0 ; MajIter mod 2 =:= 1)  -> true;  prev_neg(Term,
     MajIter,MinIter) ),
  (Term, Term =.. [_Class,Obj],
  get_object_iter(Obj,MajIter,MinIter)
        ;
  Term =.. [Class,Obj],
  lastAllModelsEntails(oid(Obj,mknf),cid(Class,mknf))).

prev_neg(Term,MajIter,MinIter) :-
  Term =..[Class,Obj], !,
  Iter1 is MinIter-1,
  tnot(allModelsEntails(oid(Obj,mknf),not(cid(Class,mknf)),MajIter,
     Iter1)).
```

On the other hand, the predicate $dlnot(Term, Iter)$ succeeds if by *closed world assumption*, the given $Term$ fails. This predicate correspond thus to how Negation as Failure is represented in our implementation for 3-valued MKNF Hybrid Knowledge Bases. However, as we described earlier, this evaluation cannot be done straightforwardly. In fact, in order to assure monotonicity, the evaluation of a $dlnot$ must take into account the result of the last outer iteration. In practical, this means that for $dlnot(Term, Iter)$ to succeed, the call $tnot(known(Term, Iter - 1, MinIter))$ has to succeed as well, i.e., the tabled predicate $known(Term, Iter - 1, MinIter)$ has to fail. Furthermore, since we are evaluating $dlnot$s by looking at the last outer iteration, we avoid both positive and negative loops, guaranteeing that the evaluation always terminates.

As described before, each outer iteration represent a variation in a T and TU model. that relates to the operators Γ and Γ' of the 3-valued MKNF Semantics definition [34]. As a result, T models are monotonically increasing whilst TU models are monotonically decreasing. To assure this monotonicity, the way $dlnot/2$ is first evaluated is very important. To assure that the first TU model is the largest model, we compel all $dlnot$s to succeed in the first outer iteration.

Afterwards, in the following iterations, $dlnot/2$ is defined as the failure of $known/3$ in the last outer iteration:

```
dlnot(_Term,0):- !.

dlnot(Term,MajIter):-
  Term =.. [Class,Obj],
        LastIter is MajIter - 1,
```

```
get_fixpoint_iter(Obj,LastIter,FinIter),
      tnot(known(Term,LastIter,FinIter)).
```

5.4.1.2 DL-Component

A given proposition is evaluated according to the ontology via a satisfiability proof using a tableau algorithm. This algorithm is implemented by CDF's theorem prover and can be called by the predicate $rec\_allModelsEntails/2$. Roughly, $rec\_allModelsEntails(Id, CE)$ succeeds if, in all models, the object Id is satisfiable with the class expression CE.

As in the rules, the given set of objects is tested in the ontology in order to infer new knowledge at each inner iteration of the algorithm. For this purpose, we define the predicate $allModelsEntails/3$ which references CDF's theorem prover at each iteration. Intuitively, $allModelsEntails(Id, CE, Iter)$ succeeds if the negation of the class expression CE is unsatisfiable, that is, if the query to CDF theorem prover: $rec\_allModelsEntails(Id, not(CE))$ fails:

```
:- table allModelsEntails/4.
allModelsEntails(Id,CE,_MajIter,_MinIter):-
        (rec_allModelsEntails(Id,not(CE)) -> fail ; true).
```

As a result, the predicate $allModelsEntails(Id, CE, MajIter, MinIter)$ succeeds if the negation of the class expression CE is unsatisfiable.

This predicate $allModelsEntails/4$ is used to perform queries, employing CDF theorem prover, for the set of objects included on the table and which belong to the domain specified by the predicate $definedClass/3$ for $Class$ and $Role$. These queries allow us to construct a model with a set of properties for a given object:

```
allModelsEntails(Object, Class(Object),Iteration)
allModelsEntails(Object, Role(Object,Object1),Iteration)
allModelsEntails(Object, not Class(Object),Iteration)
allModelsEntails(Object, not Role(Object,Object1),Iteration)
```

i.e., all the classes and roles that the object explicitly belong (or not) to.

5.4.1.3 Interaction

Earlier, we gave the idea about how the knowledge is shared from the ontology to the rules – the predicate $known(Term, MajIter, MinIter)$, which evaluates in the rules-component whether a given $Term$ is true, would also check, in the case that $Term$ does not succeed, if this $Term$ was true in the last inner iteration of the ontology model. This test is made via the predicate $lastAllModelsEntails/2$ that, roughly, goes to the last inner iteration of the ontology model and verifies if in that given iteration, $allModelsEntails/2$ succeeded for that same $Term$.

Symmetrically, we need to assure that the knowledge inferred by the rules is also passed through the ontology. To address this, we make use of a CDF feature – *cdf_intensional*. As

described before in Section 5.2, the architecture of a CDF instance can be divided into two parts – extensional facts and intensional predicates. In the extensional fact, we can define ordinary concepts, roles and facts. However, in intensional predicates one is able to define this ontology's concepts, roles and facts in a Prolog-like rule. Particularly, *cdf_intensional* allows us to define that a given $Class(Object)$ holds if it was true in the rules in the last inner iteration of the model. This is:

```
isa_int(oid(Obj,NS),cid(Class,NS1)):-
  ground(Obj),ground(Class),!,
  Call =.. [Class,Obj],
  last_known(Call).
isa_int(oid(Obj,NS),cid(Class,NS)):-
  ground(Obj),var(Class),!,
  definedClass(Call,Class,Obj),
  last_known(Call).
```

Equivalently for relations, a $Role(Object1, Object2)$ is true if it was true in the rules in the last iteration:

```
hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), ground(Role),!,
    Call =.. [Role,Obj1,Obj2],
    last_known(Call).
hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), var(Role),!,
    definedRole(Call,Role,Obj1,Obj2),
    last_known(Call).
```

The predicate $last\_known(Call)$ succeeds if, in the last inner iteration of the model obtained by the rules the $Call$ succeeded. Note that it is only possible to express unary and binary relations in the ontology, and therefore, all the n-ary terms (where $n > 2$) are ignored in the ontology.

5.4.1.4 Related Objects

A desirable property for our algorithm is that it only computes a model in the ontology for objects that are *relevant* for solving the given query. Intuitively, an object is relevant if it might be essential to the establishment of the evaluation of the query. In a top-down manner, this set of objects increases as new dependency relations between objects are disclosed. This is performed via $get\_object\_iteration/3$. For every object in $Term$, $known(Term, MajIter, MinIter)$ refers to the tabled predicate $get\_object\_Iter(Id, MajIter, MinIter)$. If the object with an Id is defined in the table, then it just returns the correspondent $MajIter$ and $MinIter$, that is, it maps to which outer and inner iteration stands the computation of Id; else, the given Id is asserted to the table and with $MajIter$ and $MinIter$ equal to 0.

Our algorithm checks for each object with an Id in the tabled $get\_object\_iteration/3$ if a fixpoint is achieved. When this happens, the given object is marked as *final*. The iteration stops

when all objects presented in the table are final, that is, when all objects have reached a fixpoint.

5.4.1.5 Iterations

Earlier we classified the iterations in two, accordingly to an inner and an outer fixpoint. An inner iteration represents the core of the computation *per se*. Here, in each iteration our algorithm derives simultaneously knowledge from the rules-component and the DL-component as described in Figure 5.5. This is done by the predicate $computeObjectPrimitives\_1/3$, which by nondeterministically calls $known/2$, and $allModelsEntails/2$ for each class and role defined by the predicate $definedClass/2$ and $definedRole/3$.

```
computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedClass(Term,_F,Obj),
  known(Term,MajIter,MinIter),
  fail.

computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedClass(_Term,Class,Obj),
  allModelsEntails(oid(Obj,mknf),cid(Class,mknf),MajIter,MinIter),
  fail.

computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedClass(_Term,Class,Obj),
  allModelsEntails(oid(Obj,mknf),not(cid(Class,mknf)),MajIter,MinIter
     ),
  fail.

computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedRole(Term,_Role,Obj,_Obj2),
  known(Term,MajIter,MinIter),
  fail.

computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedRole(_Term,Role,Obj,Obj2),
  allModelsEntails(oid(Obj,mknf),exists(rid(Role,mknf),oid(Obj2,mknf)
     ),MajIter,MinIter),
  fail.

computeObjectPrimitives_1(Obj,MajIter,MinIter):-
  definedRole(_Term,Role,Obj,Obj2),
  allModelsEntails(oid(Obj,mknf),not(exists(rid(Role,mknf),oid(Obj2,
     mknf))),MajIter,MinIter),
  fail.
```

```
computeObjectPrimitives_1(_Obj,_MajIter,_MinIter).
```

On the other hand, an outer iteration is only needed to "fix" our knowledge in order to allow a correct evaluation of $dlnot$s. Consequently, the purpose of this iteration is to call the inner iteration, which will store the result of each evaluation (to be used further by $dlnot$) via the use of XSB tabling.

After computing all the knowledge possible from a given iteration, we have to check if the object has reached an inner fixpoint, which is done by $check\_inner\_fixpoint/1$. If the latter predicate does not succeed, then the variable $MinIter$ is incremented and the algorithm tries to infer new knowledge again. Otherwise, we have to check if the object has achieved an outer fixpoint as well. If this is the case, then the object is marked in the table as $Final$ via the aid of the predicate $finalize\_object/1$. Else, the variable $MajorIter$ is incremented and the computation restored.

This description is translated to the following code:

```
computeObjectPrimitives(Id,MajIter,MinIter):-
  ((var(MajIter) ; var(MinIter)) -> get_object_iter(Id,MajIter,
     MinIter) ; true),
  computeObjectPrimitives_1(Id,MajIter,MinIter),
  checkAllObjects(Id,MajIter,MinIter),
  (check_inner_fixpoint(Id) ->
    (check_outer_fixpoint(Id) ->
       finalize_object(Id)
       ;  minor_finalize_object(Id,NewMajIter),
         computeObjectPrimitives(Id,NewMajIter,0))
         ; increment_minor_object(Id,MajIter,NewMinIter),
         computeObjectPrimitives(Id,MajIter,NewMinIter)).
```

Finally, the predicate $checkAllObjects/1$ is our way to impose that the computation only stops when all objects (defined in the table) have reached a fixpoint. Therefore, it succeeds if all the remaining objects are marked as $Final$. Otherwise, it calls $computeObjectPrimitive/3$ for these objects.

5.4.2 Usage

As in the bottom-up solution, the hybrid knowledge base must be defined by three different files in XSB and CDF located in a given folder, in order to be loaded as a whole. However, given the nature of this implementation, the content of each file is slightly different when compared to the bottom-up implementation.

- $rules.P$ – containing the set of Prolog rules and Prolog facts. A rule is defined as standard Prolog rules as follows:

  ```
  Head :- known(A_1),...,known(A_n), dlnot(B_1),...,dlnot(B_m).
  ```

Where $n, m \geq 0$, and A_n represent positive and B_m negative predicates. The predicates $known/1$ and $dlnot/1$ are thus employed by our solution for passing the computation to SLG resolution. A rule fact is defined as standard XSB facts like:

```
Head.
```

Similar to the bottom-up solution, each rule must respect DL-safeness, in order to the Hybrid Knowledge Base to be DL-safe.

- $cdf\_extensional.P$ – comprising ordinary ontology facts and concepts defined over the CDF syntax described in Section 5.2.1.1.
- $cdf\_intensional.P$ – including *intensional rules* that are defined as Prolog-like rules:

  ```
  Head :- Body.
  ```

  but where the $Head$ represents a particular ontology fact or concept that holds when the $Body$ holds as well. This $Body$ can refer to any CDF proposition or XSB library feature. However, contrary to the bottom-up implementation, $Body$ can also call back directly any (n-ary) predicate from the ontology via the use of $known/1$ and its negation with $dlnot/1$.

As an implementation of a *homogeneous* integration semantics, any atom A can be defined as a predicate, or as a proposition or both. Also, any predicate can be defined over a proposition as well as any proposition can be defined over a predicate. This results in an interchangeability between $rules.P$, $cdf\_extensional.P$ and $cdf\_intensional.P$, where any atom can be defined freely in any of these files. Inference in predicates defined in $rules.P$ is performed via SLG resolution, whilst in $cdf\_extensional.P/cdf\_intensional.P$ CDF theorem prover evaluates propositions accordingly to a tableau algorithm.

Similar to the first implementation, it is the responsibility of our algorithm to retrieve where the given A (typically needed for answer-resolution) is defined in order to evaluate correctly the given query.

5.4.3 Examples

Next, we continue by presenting some examples to illustrate the behaviour of our algorithm.

Example 5.7. *Consider the following DL-safe Hybrid Knowledge Base and the query* $one(X)$.

$$\mathbf{K}\ one(X) \leftarrow \mathbf{K}\ p(X), \mathbf{not}\ two(X).$$
$$\mathbf{K}\ two(X) \leftarrow \mathbf{K}\ p(X), \mathbf{not}\ one(X).$$
$$p(obj).$$
$$\neg Two(obj)$$

This hybrid knowledge base is translated to the following syntax in XSB.

```
one(X) :- known(p(X)), dlnot(two(X)).
two(X) :- known(p(X)), dlnot(one(X)).
p(obj).
```

And to the following syntax in CDF.

```
necessCond_ext(oid(obj,mknf),vid(not(cid(two,mknf)))).
```

In order to answer to the query, our solution starts to call the rule $one(X)$. This calls p(X) which grounds X to obj, adding it to the table, by asserting the predicate $get\_object\_iter(obj, 0, 0)$. Afterwards, it computes iteratively a model for obj, following the Definition 5.10 and considering all its properties and relations until a fixpoint is achieved. Subsequently, it will continue by performing this computational until an outer fixpoint is accomplished.

Next, we give a trace of the evaluation of the query $one(obj)$, by describing what and why is inferred in each outer iteration.

Iter = 0 *As a consequence of our special evaluation of $dlnots$ in the two first iterations, in this first outer iteration, since we are computing the larger* optimistic *model, all $dlnots$ succeed. Yet, as we are computing a TU model, an atom A does not succeed in the case that $\neg A$ is represented in the ontology. Hence, from the evaluation of $\neg Two(obj)$ the predicate $known(two(obj), 0, final)$ fails. On the other hand, since $p(obj)$ holds and all $dlnots$ are forced to succeed, $known(one(obj), 0, final)$ succeeds as well.*

$$M_0 = \{known(one(obj), 0, final), known(p(obj), 0, final), \\ allModelsEntails(oid(obj, mknf), notcid(two, mknf), 0, final)\}$$

Iter = 1 *The result from the previous iteration is used for the evaluation of $dlnots$. As a result, $dlnot(one(obj), 1)$ fails as $known(one(obj), 1)$ holds in the last model. Symmetrically, $dlnot(two(obj), 1)$ succeeds as $tnot(known(two(obj), 0, final)$ succeeds as well. Accordingly, $known(one(obj), 1, final)$ and $known(p(obj), 1, final)$ holds in the rules, and the ABox: $\neg Two(obj)$ is inferred by the ontology*

$$M_1 = \{known(one(obj), 1, final), known(p(obj), 1, final), \\ allModelsEntails(oid(obj, mknf), notcid(two, mknf), 1, final)\}$$

Iter = 2 *This latter iteration is only needed for a verification of the fixpoint. In fact, the call $dlnot(two(obj), 2)$ still fails as $known(two(obj), 2, final)$ is derived in the last iteration. From this, the predicate $known(one(obj), 2, final)$ continues to hold. On the other hand, $known(p(obj), 2, final)$ and $\neg Two(obj)$ succeed as they are stated as facts in our knowledge base.*

$$M_2 = \{known(one(obj), 2, final), known(p(obj), 2, final), \\ allModelsEntails(oid(obj, mknf), notcid(two, mknf), 2, final)\}$$

Finally, we look to what is obtained in our models to decide the result of the query $one(X)$. *Since* $known(one(obj)$ *holds, our solution returns true for* $X = obj$.

Example 5.8. *Next, we present an example with multiple objects given the query* $third(p1)$*:*

```
first(p1).
third(p1):- known(second(p2)), known(first(p1)).).

isa_ext(cid(second,mknf),cid(p2,mknf)).
```

The algorithm starts by adding $get\_object\_iter(p1, 0, 0)$, *computing all its properties. However, the evaluation of* $known(third(p1))$ *is depends on the value of* $known(second(p2))$. *Since our algorithm does not know* $p2$, *it is added to the table by asserting* $get\_object\_iter(p2, 0, 0)$. *Consequently, the solution will start to evaluate* $p2$ *accordingly to its properties and the computation only stops until* $p2$ *has reached a fixpoint as well. Thus, we force the algorithm to derive* $Second(p2)$ *from the ontology that will make* $known(second(p2))$ *to succeed. As a result,* $known(third(p1))$ *succeeds as well.*

Since the knowledge base is positive, all this is inferred in the first iteration. The latter two iterations are needed to assure that the fixpoint is achieved.

$$\begin{aligned} M_0 = M_1 = M_3 = \{&known(first(p1), \_, final), known(third(p1), \_, final),\\ &known(second(p2), \_, final),\\ &allModelsEntails(oid(p1, mknf), cid(first, mknf), \_, final),\\ &allModelsEntails(oid(p1, mknf), cid(third, mknf), \_, final),\\ &allModelsEntails(oid(p1, mknf), cid(first, mknf), \_, final)\} \end{aligned}$$

5.4.4 Discussion and validation

In [3] it is presented a goal-driven procedure for Hybrid MKNF knowledge bases, which capture reasoning within the ontology, as an interaction with an external oracle. This oracle assumes all rule literals necessary to the computation of a proposition to be true, returning the computation to the SLG-based rule engine by passing a set with all possible sets of rule-literals that, if assumed true, prove the given query. It is then the task of the rule engine to check whether any of these possibilities is the case, without the need of any further call to the DL-reasoner for the same query. However, this procedure has some drawbacks: if the first rule-atom found by the DL-reasoner happens to be false, the computation will continue until the DL-reasoner finds all (possibly too many) possibilities, just to fail afterwards when the rule engine takes the lead again.

For our solution, this restriction is overcome giving the particular integration between the rule engine (XSB) and the ontology framework (CDF). In fact, CDF is implemented over XSB and, therefore, it enjoys from most of XSB properties, as the ability to define rules and tabling. As a result, CDF can access directly to tabled XSB predicates, as well as their residual.

Nevertheless, avoiding the excess of computation of the DL-reasoner comes at the price of renouncing to a pure goal-driven procedure. If by one hand, when reasoning in Description Logics via the use of tableaux procedures, the construction of the model is essential, on the other hand, our evaluation of a predicate on the rules-component is not completely top-down. As we discussed in Section 5.4.1.1, default negations are evaluated in accordance to the model inferred in the last outer iteration. This holds a need for computing a second fixpoint, and to decide the truth value of $dlnot$s similar to the bottom-up fashion way.

Yet, it is this resemblance to the definition of the bottom-up approach, the underpinning of our idea of correction. In fact, even though we have not presented a formal proof of soundness and completeness for our algorithm, we gave along the description an informal intuition of its correction by referring to a similarity between the two implementations. Particularly, it is undeniable the correspondence between the core of our computation represented in Definition 5.10 and the Definition 4.15 from the 3-valued MKNF definition, and between the operators Γ / Γ' and our inner / outer fixpoints. As a result, one can define our goal-driven implementation as an optimisation of the bottom-up approach where the computation is limited to the set of *relevant* objects, and the evaluation of positive predicates is performed by the use of SLG resolution..

Still, at this moment, not much work exists in providing concrete applications for reasoning in hybrid knowledge bases that comprise rules and ontologies. Implementations like dlvhex [52] are able to reason in hybrid knowledge bases, but always limiting the interaction between the two components. Consequently, our implementation is the first implementation where rules and ontologies are interleaved and interact with no restrictions, leaving us without a point of comparison. Even so, inspired by the results of the bottom-up implementation, we developed an exhaustive test suit to reinforce the robustness of our solution. This implementation along with a fragment of the test suit is available for download at the XSB CVS repository and will possibly be integrated as a package in the next XSB release.

6 . Conclusions

Motivated primarily by the demand to complete the Semantic Web architecture with the integration of monotonic ontologies and nonmonotonic rule languages, we developed two novel implementations for reasoning in hybrid knowledge bases that are accordant to the MKNF Well-Founded Semantics.

Moreover, implementations with such characteristics are not only useful in the Semantic Web context. In fact, several use case applications have been presented [19], [47], [58], [6] with requirements for a close and safe combination between ontologies and rules.

Contrary to other implementations developed so far, our implementations are able to handle hybrid knowledge bases where rules and ontologies are fully integrated, that is, the combination is such that both the rules can refer to predicates in the ontology, as the ontology can refer to predicates defined in the rules. However, this combination is far from being trivial – since we are combining two completely different systems, as rules employ closed world assumption, whereas ontologies adhere to open world assumption, negation (both explicit and default) cannot be treated straightforwardly. Nevertheless, given the properties of the semantics employed, the implementations are able to preserve the complexity of CDF Description Logics, maintaining tractability for Type-0 instances, and `EXPTime`-complete for Type-1 instances.

The first implementation we presented herein refers to a bottom-up implementation mainly based on the application of the operators defined in the MKNF Well-Founded Semantics. Still, our solution extends the definition of the semantics by allowing the programmer to define non-ground rules, as long as they respect the condition of DL-Safeness. As a bottom-up implementation, it works by applying rules and entailment to the given facts and ABoxes to infer new facts, repeating this process until no more facts are derivable, i.e., when a fixpoint is accomplished. In the end, it computes the whole set of predicates and propositions that are true, false and undefined, concerning a given hybrid knowledge base. This approach is particularly efficient for small and uniform hybrid knowledge bases as redundant derivations are avoided.

On the other hand, the goal-driven implementation makes use of XSB's SLG Resolution together with CDF's theorem prover to answer a specific query. From this, one can see that the gist of this second implementation is slightly different – being a goal-driven implementation, it is query-oriented, only computing predicates and propositions of a given object as they are requested by the algorithm. Hence, this latter implementation assumes that not everything needs to be computed in order to answer a query, which makes it more suitable for the distributed nature and immeasurable size of the Web.

In the beginning of this report we presented a wide set of approaches in the matter of combining rules and ontologies and, up until now, not much work exists in providing practical implementations for these approaches. Our solutions intend to reduce this gap between theoretical proposals and tangible applications. We believe that our two implementations represent a relevant contribution, first by achieving an important milestone in the Semantic Web architecture layer, and second by paving the way for the arising of several applications that desire to

reap the benefits of such combination.

For future work, an important challenge is to abstract the implementation to handle any given DL reasoner. Currently, our solution suffers from embracing two systems that are roughly the same. It would be interesting to present an implementation that could cope with popular DL reasoners like KAON2[1], Pellet[2] or Fact++[3].

Another riveting challenge is to enhance the performance of the CDF theorem prover by restricting the set of propositions to be computed. For instance, this could be accomplished by building a dependency graph between the predicates and propositions of the hybrid knowledge base.

These two challenges lead to research in two opposite directions – the first seeks to define an abstract framework in order to handle multiple DL reasoners, whilst the latter needs to exploit of the specifications of CDF theorem prover to minimise the computational power.

[1] http://kaon2.semanticweb.org/
[2] http://clarkparsia.com/pellet/
[3] http://owl.man.ac.uk/factplusplus/

Bibliography

[1] José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira. Slx – a top-down derivation procedure for programs with explicit negation. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 424–438, Cambridge, MA, USA, 1994. MIT Press.

[2] José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira. A logic programming system for nonmonotonic reasoning. *J. Autom. Reasoning*, 14(1):93–147, 1995.

[3] José Júlio Alferes, Matthias Knorr, and Terrance Swift. Queries to hybrid mknf knowledge bases through oracular tabling. *To Appear*, 2009.

[4] José Júlio Alferes and João Alexandre Leite, editors. *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*. Springer, 2004.

[5] José Júlio Alferes and Luís Moniz Pereira. Reasoning with logic programming. In *LNAI*. Springer Verlag, 1996.

[6] Jürgen Angele and Michael Gesmann. Data integration using semantic technology: A use case. *Rules and Rule Markup Languages for the Semantic Web, International Conference on*, 0:58–66, 2006.

[7] Grigoris Antoniou, Carlos Viegas Damásio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter F. Patel-Schneider. Combining Rules and Ontologies. A survey., 2005.

[8] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[9] Franz Baader, Diego Calvanese, Deborah L. Mcguinness, Daniele Nardi, and Peter F. Patel-Schneider. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[10] Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors. *Reasoning Web, Second International Summer School 2006, Lisbon, Portugal, September 4-8, 2006, Tutorial Lectures*, volume 4126 of *Lecture Notes in Computer Science*. Springer, 2006.

[11] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Journal of Logic Programming*, pages 269–283, 1987.

[12] Harold Boley and M. Kifer. The rule interchange format: An interim report. *The Association of Logic Programming Newsletter*, 20:215–232, 2007.

[13] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Al-log: integrating datalog and description logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.

[14] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Axel Polleres. Rules and ontologies for the semantic web. In *Reasoning Web*, pages 1–53, 2008.

[15] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *In Proc. IJCAI-2005*, pages 90–96. Professional Book, 2005.

[16] Jocelyne Faddoul, Nasim Farsinia, Volker Haarslev, and Ralf Möller. A hybrid tableau algorithm for alcq. In *Description Logics*, 2008.

[17] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

[18] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[19] Christine Golbreich, Olivier Bierlaire, Olivier Dameron, and Bernard Gibaud. Use case: Ontology with rules for identifying brain anatomical structures. In *Rule Languages for Interoperability*, 2005.

[20] Bernardo Cuenca Grau. Owl 1.1 web ontology language, tractable fragments, 2007.

[21] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Journal of Web Semantics*, 2008. To Appear.

[22] Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, Peter Patel-schneider, and Ulrike Sattler. Next steps for owl. In *In OWL Experienced and Directions*, 2006.

[23] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[24] RIF Working Group. `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`.

[25] RIF Working Group. Creating rif dialects: The rif architecture. Technical report, W3C Working Draft, `http://www.w3.org/2005/rules/wg/wiki/Arch`, 2007.

[26] The XSB Resource Group. `http://xsb.sourceforge.net/`.

[27] Jeff Heflin. Web ontology language (owl) use cases and requirements, 2003.

[28] Jeff Heflin, Raphael Volz, and Jonathan Dale. Requirements for a web ontology language, 2002.

[29] Ian Horrocks. Semantic web: the story so far. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 120–125, New York, NY, USA, 2007. ACM.

[30] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. 10th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, June 2006.

[31] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml, 2004.

[32] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 17–29. Springer, 2003.

[33] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1990.

[34] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. A well-founded semantics for hybrid mknf knowledge bases. In *Description Logics*, 2007.

[35] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. A coherent well-founded model for hybrid mknf knowledge bases (extended version). In *ECAI*, pages 99–103, 2008.

[36] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Elp: Tractable rules for owl 2. In *International Semantic Web Conference*, pages 649–664, 2008.

[37] Vladimir Lifschitz. Nonmonotonic databases and epistemic queries. In *IJCAI*, pages 381–386, 1991.

[38] Vladimir Lifschitz. Minimal belief and negation as failure. *Artificial Intelligence*, 70:53–72, 1994.

[39] Francesca A. Lisi and Donato Malerba. Ideal refinement of descriptions in al-log. In *Inductive Logic Programming*, volume 2835 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2003.

[40] John W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

[41] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.

[42] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and logic programming live together happily ever after? In *Proc. of the 5th International Semantic Web Conference (ISWC 2007)*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006.

[43] Boris Motik and Riccardo Rosati. Closing Semantic Web Ontologies. Technical report, University of Manchester, UK, 2006.

[44] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.

[45] Ray Reiter. A logic for default reasoning. *Artificial Intelligence*, pages 68–93, 1987.

[46] Ray Reiter. On closed world data bases. *Artificial Intelligence*, pages 300–310, 1987.

[47] Jeffrey Ritter. The use of rule languages for policy and legal compliance. In *Rule Languages for Interoperability*, 2005.

[48] Riccardo Rosati. Towards expressive kr systems integrating datalog and description logics: preliminary report. In *In Proc. DL-1999*, pages 160–164, 1999.

[49] Riccardo Rosati. Towards expressive kr systems integrating datalog and description logics: preliminary report. In *Description Logics*, 1999.

[50] Riccardo Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In *KR-06*, pages 68–78, 2006.

[51] Konstantinos Sagonas, Terrance Swift, and David S. Warren. Xsb as an efficient deductive database engine. In *In Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 442–453. ACM Press, 1994.

[52] Roman Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, Technischen Universitaet Wien Fakultaet fuer Informatik, Glockengasse 6/19, A-1020 Wien, 2006.

[53] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[54] Terrance Swift. A new formulation of tabled resolution with delay. In *In Recent Advances in Artifiial Intelligence*, pages 163–177. Springer-Verlag, 1999.

[55] Terrance Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240, 1999.

[56] Terrance Swift. A note on trailing in the slg-wam. Technical report, Available at `http://www.cs.sunysb.edu/~tswift/`, 2001.

[57] Terrance Swift and David S. Warren. *Coherent Description Framework Programmer's Manual*. XSB Inc.

[58] Kjell Tangen. Large-scale use and exchange of classification rules in the maritime industry. In *Rule Languages for Interoperability*, 2005.

[59] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[60] Herman J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):79–115, October 2005.

[61] David S. Warren. Memoing for logic programs. *Commun. ACM*, 35(3):93–111, 1992.