Foundations and Applications of Knowledge Representation for Structured Entities

Despoina Magka

Lincoln College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2013

Contents

List of Figures

List of Tables

Abstract

Description Logics form a family of powerful ontology languages widely used by academics and industry experts to capture and intelligently manage knowledge about the world. A key advantage of Description Logics is their amenability to automated reasoning that enables the deduction of knowledge that has not been explicitly stated. However, in order to ensure decidability of automated reasoning algorithms, suitable restrictions are usually enforced on the shape of structures that are expressible using Description Logics. As a consequence, Description Logics fall short of expressive power when it comes to representing cyclic structures, which abound in life sciences and other disciplines.

The objective of this thesis is to explore ontology languages that are better suited for the representation of structured objects. It is suggested that an alternative approach which relies on *nonmonotonic existential rules* can provide a promising candidate for modelling such domains.

To this end, we have built a comprehensive theoretical and practical framework for the representation of structured entities along with a surface syntax designed to allow the creation of ontological descriptions in an intuitive way. Our formalism is based on nonmonotonic existential rules and exhibits a favourable balance between expressive power and computational as well as empirical tractability. In order to ensure decidability of reasoning, we introduce a number of acyclicity criteria that strictly generalise many of the existing ones. We also present a novel stratification condition that properly extends 'classical' stratification and allows for capturing both definitional and conditional aspects of complex structures. The applicability of our formalism is supported by a prototypical implementation, which is based on an off-the-shelf answer set solver and is tested over a realistic knowledge base. Our experimental results demonstrate improvement of up to three orders of magnitude

in comparison with previous evaluation efforts and also expose numerous modelling errors of a manually curated biochemical knowledge base.

Overall, we believe that our work lays the practical and theoretical foundations of an ontology language that is well-suited for the representation of structured objects. From a modelling point of view, our approach could stimulate the adoption of a different and expressive reasoning paradigm for which robustly engineered mature reasoners are available; it could thus pave the way for the representation of a broader spectrum of knowledge. At the same time, our theoretical contributions reveal useful insights into logic-based knowledge representation and reasoning. Therefore, our results should be of value to ontology engineers and knowledge representation researchers alike.

Acknowledgements

I would like to express my sincerest thanks to:

- Prof. Ian Horrocks for being an inspiring teacher, a generous supervisor and an impeccable researcher. This thesis would not have been possible without the brilliance of his ideas and remarks.

- Dr Markus Krötzsch for his rigorous scholarship and academic ethos. I am grateful to him for consenting to undertake me under his supervision and for putting all this enthusiasm into it.

- Dr Yevgeny Kazakov, my master's thesis advisor, for initiating me to the art of writing proofs, formalising arguments and selecting sensible research topics.

- Current and former members of the Oxford KRR group Yavor Nenov, Ernesto Jimenez-Ruiz, Ana Armas, Bernardo Cuenca Grau, Robert Piro, Zhe Wang and Clemens Kupke for their delightful company and their advice in all sorts of 'ontological' matters; Karen Barnes for her warm presence in the office during the last months of my DPhil; fellow student and Lincoln member Jan Botha for sharing with me all the joys and sorrows of doing a DPhil in the comlab.

- My friends in Oxford Drs Bruno Marnette, Evangelia Kouyioumoutzi, Nikos Tzevelekos, Mehrnoosh Sadrzadeh, Hristina Palikareva, Anna Lefteratou, Maria Cartolano, Bjorn Pieper and Chris Batchelor-McAuley for sharing fun moments with me and offering valuable advice about research; I also thank Chris for answering my chemistry question. I am very grateful to my parents, Giannis and Katerina, for their faith and support.

- Dr Giorgos Stoilos for being a most valuable officemate—always ready to offer hotline support when needed; Dr Giorgos Stamou for introducing me to the

fascinating field of logic-based knowledge representation and forming with the previous Giorgos a fantastic (officemate) duo.

- Janna Hastings, Dr Michel Dumontier and Prof. Sebastian Rudolph for fruitful discussions about knowledge representation.

- Dr Sirichai Chongchitnan for being an invaluable college advisor; Carmella Elan-Gaston, the Lincoln College Graduates Officer, for her precious support; the Lincoln College kitchen staff for hundreds of tasty lunches at the Hall.

- EPSRC for funding my doctoral studies and the IJCAR'10, OWLED'12, DL'12, IJCAI'13 organising committees for providing travel grants; I also acknowledge BioMed Central for financing the SWAT4LS'12 best paper prize.

Chapter 1

Introduction

Description Logics form a family of powerful ontology languages widely used by researchers and industry experts to capture and intelligently manage knowledge about the world. A key advantage of Description Logics is their amenability to automated reasoning that enables the deduction of not explicitly stated knowledge. However, in order to ensure decidability of automated reasoning algorithms, suitable restrictions are usually enforced on the shape of structures that are expressible using Description Logics. As a consequence, Description Logics fall short of expressive power when it comes to representing cyclic structures, which abound in life sciences and other disciplines.

The objective of this thesis is to explore ontology languages that are are better suited for the representation of objects with cyclic structure. It is suggested that an alternative approach which relies on *nonmonotonic existential rules* can provide a promising candidate for modelling such domains. This is demonstrated by the definition of an ontological framework for the representation of structured entities that builds upon nonmonotonic existential rules and that exhibits an encouraging balance between expressive power and computational as well as empirical tractability.

1.1 Logical Background of Knowledge Representation

Knowledge representation technologies play a substantial role in intelligently managing the rapidly growing amount of information readily available nowadays. Logic lies at the heart of modern knowledge representation systems as it provides the formal underpinnings of numerous *ontology languages*. Ontology languages are formalisms extensively used by researchers and industry experts to store human knowledge in *knowledge bases*, which are machine-processable information repositories. A key advantage of knowledge bases is their amenability to automated reasoning that enables derivation of implicitly stated information by means of logical inference. However, increasing the expressive power of ontology languages can come at a significant computational price for the corresponding reasoning algorithms. Therefore, achieving a favourable balance between expressiveness and tractability is one of the most pursued questions within ontology language design and one of the main focuses of this work.

The beginning of practical knowledge representation can be traced back to the early 1970s when artificial intelligence researchers developed the first expert systems such as DENDRAL [38] and MYCIN [216] to perform elucidation of molecular structures and diagnosis of blood diseases, respectively. These systems had the distinguishing advantage of relying on a separate knowledge base that could be updated and extended with new information while keeping intact the code for interpreting and using that knowledge. Since then, extensive research has been carried out throughout several generations of knowledge representation systems with many efforts concentrated on the design of suitable formalisms for creating knowledge bases. *Description logics* and *rule-based formalisms* are two of the major language paradigms that have shaped logic-based knowledge representation over the past few decades. We provide a high-level overview of foundations and applications of these logics next.

1.1.1 Description Logics

Description logics (DLs) form a family of ontology languages with well-defined syntax and semantics, so that reasoning tasks can be automated using logical deduction

[12]. Ontology engineers can thus build DL knowledge bases to formalise knowledge domains. One can create elements of a DL knowledge base by combining the main DL constructs and a set of logical connectives in expressions valid according to the DL syntax; the main DL constructs are *concepts*, *roles* and *individuals* while logical connectives include symbols, such as conjunction ($\sqcap$), disjunction ($\sqcup$) or negation ($\neg$), depending on the specific expressivity of each DL variant. In a nutshell, concepts are used to describe entities with common characteristics and roles are used to associate pairs of these entities with relationships, while individuals are used to describe instances of concepts. For example, Panda is a concept capturing all the animals belonging to that species, and Bamboo is a concept describing the eponymous plants. Similarly, mostlyEats is an example of a role associating entities with their main diet component. One can combine concepts and roles with logical constructors to compose concept expressions, such as $\exists$mostlyEats.Bamboo, which denotes all entities that mainly feed on bamboo. A DL knowledge base usually consists of two components, a TBox that contains *concept inclusions* and an ABox that contains *assertions*. A concept inclusion is a subclass/superclass relation between concepts, e.g.

$$\mathsf{Panda} \sqsubseteq \exists \mathsf{mostlyEats.Bamboo}$$

is a concept inclusion which specifies that pandas mainly feed on bamboo. Similarly, if Plant and Herbivore are concepts describing plants and herbivorous organisms respectively, the following is an example of a concept inclusion, stating that everything that has plant as its main diet component is a herbivore.

$$\exists \mathsf{mostlyEats.Plant} \sqsubseteq \mathsf{Herbivore}$$

On the other hand, an assertion is a statement that relates individuals with concepts or pairs of individuals with roles; for example, if chichi is an individual, Panda(chichi) is an assertion declaring the entity denoted by chichi to be a panda.

A significant strength of DLs is the availability of reasoning algorithms that are able to automatically check the consistency of DL knowledge bases and deduce knowledge that has not been explicitly stated. This is due to the formally specified seman-

tics of DLs that strictly define the justifiably entailed statements. For example, if a DL knowledge base contains the previously mentioned concept inclusions and the concept inclusion $\mathsf{Bamboo} \sqsubseteq \mathsf{Plant}$, which states that bamboo is a kind of plant, then a reasoning algorithm can discover that

$$\mathsf{Panda} \sqsubseteq \mathsf{Herbivore}$$

because it logically follows from the previously described axioms. Importantly, it can be proved that many of the available reasoning algorithms are *sound*, *complete* and *terminating*, which guarantees that every derived subsumption is correct, that every correct subsumption is derived and that the derivation process always terminates, respectively. A useful application of a reasoning algorithm is that it can be used to *classify* a knowledge base, that is to automatically construct the complete hierarchy of concepts induced by the TBox concept inclusions.

DLs can also be seen as decidable fragments of first-order logic. In particular, DL semantics follows the first-order logic convention and adopts the open-world assumption according to which missing information is treated as *not known* rather than *false*. For example, if one replaces the axiom $\exists\mathsf{mostlyEats}.\mathsf{Plant} \sqsubseteq \mathsf{Herbivore}$ with $\exists\mathsf{mostlyEats}.(\neg\mathsf{Meat}) \sqsubseteq \mathsf{Herbivore}$, then the subsumption $\mathsf{Panda} \sqsubseteq \mathsf{Herbivore}$ is no longer derivable, as that would require to explicitly state that Plant and Meat are disjoint concepts.

Historically, one of the early occurrences of DLs in knowledge representation was the KL-ONE language [32] and its implementation NIKL [183] in the beginning of the 1980s, although the subsumption problem for KL-ONE was eventually proved to be undecidable [214]. Following KL-ONE, a wide range of DL variants and corresponding systems were proposed, a comprehensive overview of which is provided by Woods and Schmolze [248]. In the mid-1990s the GRAIL system [205] was designed and implemented to support reasoning for the GALEN ontology [203], a large medical terminology aiming to provide a common hierarchical schema which could be used by application designers. However, it was soon discovered that GRAIL's subsumption reasoning was incomplete [124] which led to the development of a new DL system

to replace it, the FaCT reasoner [125]. FaCT was one of the first systems to implement a sound and complete classification algorithm for a relatively expressive DL and at the same time to behave well in practice [126]. Following that, a significant amount of research was invested in continuously improving the theories and systems of DLs, which sparked more interest in DLs by ontology engineers. As a result, DLs were established as the formal foundations of the *Web Ontology Language*, which in 2004 became the W3C standard ontology language for representation of World Wide Web objects [130, 23]. The DL $\mathcal{SHOIN}(D)$ was selected as the logical underlying formalism of OWL, the subsumption problem for which is NExpTime-complete [229]. A wide range of OWL conformant and efficient reasoners were thus developed such as Fact++ [231], RacerPro [107], Pellet [219] and HermiT [95]. A stream of feature requests in the subsequent years resulted in the standardisation of a more expressive version of OWL in 2009—OWL 2 [63, 193], which drew upon the DL $\mathcal{SROIQ}(D)$, the subsumption problem for which is N2ExpTime-complete [127]. Since then, reasoners for (subsets of) OWL 2 are being constantly developed, examples of which include Owlgres [226], CB [136], ELK [137] and MoRe [207].[1]

One of the major application of DLs has been the automatic construction of hierarchies for life science applications. This is a direct consequence of the numerous benefits that taxonomical structures bear, such as the ability to easily navigate through hierarchically organised information or to extract statistical similarity measures by processing the taxonomy tree [84]. Moreover, OWL bio-ontologies can provide a framework to catalogue, represent and index bioscience data—which is currently being generated at an astonishing rate—by acting as common reference terminologies; prominent OWL life science vocabularies include the GALEN project (23,141 concepts as of January 2007 [204]), the Gene Ontology (39,311 concepts as of May 2013 [52]), the National Cancer Institute Ontology (97,547 concepts as of April 2013 [218]) and SNOMED CT (395,036 concepts covering clinical knowledge as of February 2013 [60]).[2] Fields where DL-based technologies have been used in the ways described

[1] A list of OWL reasoners is maintained by the W3C OWL Working Group at `http://www.w3.org/2007/OWL/wiki/Implementations`.

[2] A catalogue of life sciences OWL ontologies can be found at BioPortal [190].

above for biological knowledge management include but are not limited to:

- drug discovery [242, 250];
- biomedical information integration [61, 244, 119, 74];
- cheminformatics [58, 243, 55, 56, 110];
- biochemical pathways [118];
- proteomics [245, 247];
- genomics [117, 120, 93];
- neuroanatomy [192, 182].

Other examples of application areas for DLs include grid computing [198], optics [187], geoscience [86], natural language processing [235], multimedia [224], cultural heritage [151] and the so-called Semantic Web [81].

1.1.2 Rule-Based Formalisms

Rule-based formalisms originated with logic programming, which emerged in the early 1970s as a new programming paradigm that promoted a declarative approach [160]. The main modelling constructs of logic programs are *rules* and *facts*. Rules can be thought as conditional sentences consisting of a premise and a conclusion, such that if the conditions specified by the premise are true for a certain configuration, then the conclusion must also hold for the same configuration. Rules resemble TBox concept inclusions in that they conceptually model some aspect of the world. In logic programming, a rule is formalised as a first-order logic Horn implication where function symbols may occur. Facts on the other hand are similar to ABox assertions and are used to capture concrete instances of the world. Although the initial motivation for logic programming was natural language processing, later applications of the language spanned a wide range of fields such as computer-aided manufacturing, software engineering, transportation and telecommunication [69]. In the late 1980s, the *stable model semantics* was introduced for logic programming [92]. Since then

the focus of logic programming applications has shifted towards solving difficult (i.e. NP-hard) search problems, which gave birth to the area of answer set programming [48]. In the mid-1990s, Prolog, the principal language for logic programming became an ISO standard [69]. Unlike OWL, reasoning tasks for Prolog are undecidable, i.e. posing a query to a Prolog program may result to a non-terminating computation or an incomplete answer.

Logic programming also gave rise to *datalog*, a purely declarative language whose purpose was to serve as a query language for deductive databases [4, 50]. Similarly to logic programming, datalog programs consist of facts and rules; however, datalog rules are function-free and have an additional safety restriction, which prohibits occurrence in the head of variables that do not occur in the body. These limitations render unnecessary the examination of (potentially infinite) fresh terms during reasoning and provide a way to establish decidability. An example of a datalog rule is the following, which specifies a sufficient condition for a bear to be a panda:

$$\mathsf{bear}(\mathsf{x}) \wedge \mathsf{hasPart}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{hasPart}(\mathsf{x}, \mathsf{z})$$
$$\wedge\, \mathsf{blackFur}(\mathsf{y}) \wedge \mathsf{eyes}(\mathsf{z}) \wedge \mathsf{isAround}(\mathsf{y}, \mathsf{z}) \rightarrow \mathsf{panda}(\mathsf{x})$$

The rule above encodes the following natural language statement: every bear that has black fur around its eyes is a panda. The main reasoning tasks over datalog programs are ExpTime-complete w.r.t. the number of the rules and P-complete w.r.t. the number of the facts [65]. Reasoning becomes harder for more expressive variants of datalog, such as the ones that allow for negated atoms in the body. Many optimisation techniques have been developed for evaluation of datalog programs [4] which makes datalog reasoners highly efficient in practice. Datalog formalisms have thus been applied to a number of domains including declarative networking [161], program analysis [33], security [133] and web data extraction [97]; Huang et al. provide a survey on datalog applications [131].

Similarly to DLs, pure datalog can be seen as a decidable fragment of first-order logic; in fact, there is a significant overlap between the expressive power of datalog and various DL languages. However, there are still statements that can

be formulated in only one of the two formalisms. For example, the DL axiom $\mathsf{Panda} \sqsubseteq \exists \mathsf{mostlyEats.Bamboo}$ cannot be expressed in plain datalog as this would require the occurrence in the head of an existentially quantified variable. At the same time, the panda recognition datalog rule above does not have an equivalent DL formulation as most DLs preclude the description of structures containing cycles. As a consequence, numerous combinations of datalog-based and DL-based languages have been investigated in an effort to exploit the strengths of both, representative examples of which include Description Logic Programs [105], the Semantic Web Rule Language (SWRL) [129] and DL-safe rules [186]. However, these hybrid formalisms usually result in undecidability or significant limitations in expressivity. In the same vein, various extensions of datalog have appeared recently, such as *Datalog*$^{\pm}$ [42] or *existential rules* [17], with the objective to capture expressive features typical of DLs—mainly existentially quantified variables in the head. A notable application of Datalog$^{\pm}$ is extraction of relevant information over unstructured web data [87].

1.2 Modelling Beyond Tree-Shapedness

Albeit very successful in the representation of various domains, DLs exhibit a fundamental inability to faithfully represent cycles. This is a consequence of a generalised *tree-model property* characterising DLs, according to which each DL knowledge base is satisfied by at least one tree-like interpretation. This property accounts for the robust computational properties of DLs [233], but at the same time hinders the description of non-tree-like structured entities, such as physically composed objects that abound in life sciences and other disciplines. As a consequence, these entities tend to be modelled in an approximate way, which results in missing subsumptions and, thus, incomplete hierarchies.

A typical example of a domain where DLs fall short of expressive power when it comes to representing its objects is the chemical domain. Since molecular structures are often highly cyclic, chemical ontology engineers can only provide underconstrained DL descriptions of them which lead to poor taxonomies of the chemical space. This

restriction became particularly apparent within the development of ChEBI, one of the most established biochemical ontologies created and maintained by the European Bioinformatics Institute (EBI) [111]. Due to the limitations outlined above, the ChEBI ontology engineers failed to model molecular graphs in the ontology level, excluding thus the main content of ChEBI from logical reasoning and delegating the tedious task of chemical classification to biocurators.

Besides the chemical domain, the inadequacy of DLs to express cycles has manifested itself in a variety of contexts and applications [164]. For instance, in the area of mechanical engineering one often needs to describe objects whose parts are arranged in a cyclic way, such as for example a vehicle that has an engine part and a tank part where the two parts are linked together [102]. For analogous reasons, human anatomy structures, e.g. the human heart that consists of four interconnected chambers, cannot be modelled with sufficient precision either [184]. A similar case emerges from the software engineering domain where one needs to identify design patterns with an intricate non-tree structure from source code [6]. Further examples involve event recognition applications, where parthood and temporal relations intermingle into cyclic patterns [241], and logic-based representation of legal knowledge where diamond-shaped structures are omnipresent as they encode the core notion of a transaction [121].

As we saw earlier, a number of hybrid formalisms combining DLs and rule-based modelling has been suggested in order to address this lack of expressive power. SWRL rules [129] extend DL knowledge bases with rules that can encode structured entities; nevertheless, adding rules to even a subset of OWL makes the basic reasoning problems undecidable [155]. One can ensure decidability by imposing suitable restrictions with DL-safe rules [186]; however, this solves the problem only partially since it restricts inferences to individuals explicitly named in the ABox instead of classifying the conceptual axiomatisation of structures. In order to remedy this effect, a Description Graphs (DGs) framework was proposed that encompasses DLs and rules for describing non-tree-like structures. Nonetheless, the DGs approach enabled derivation of only some of the subsumptions missing from the taxonomy and introduced other

limitations, such as restricted use of roles and an impractical acyclicity condition (a more detailed technical discussion with examples appears later in Section 3.1.3).

Furthermore, while assessing the suitability of the DGs formalism and through discussions with EBI ontology engineers, we discovered that the open-world assumption—which is in principle adopted by DLs—complicates the modelling of classes that are essential to perform structure-based classification. In fact, it is much easier to define classes that depend on the absence of certain characteristics (such as objects that lack certain kinds of parts) with nonmonotonic negation, which—unlike first-order logic negation—assumes information that does not occur in the knowledge base to be false.

Given the expressivity requirements outlined above, our hypothesis is that datalog rules with existentials in the head and nonmonotonic negation in the body are a promising candidate for the representation of complex structures. Our claim is substantiated by the availability of numerous state-of-the-art datalog engines that can be exploited for reasoning over such rules [153, 89]. The scope of this work is to investigate the theoretical feasibility and assess the practical usability of an ontology language based on nonmonotonic existential rules for the representation of structured objects.

1.3 Key Features of an Ontology Language

In the previous section, we expressed our intention to explore the potential of nonmonotonic existential rules as an ontology language for structured domains. In this section, we list what we consider to be the core requirements of an ontology language.

- The reasoning problem (computation of subsumptions in our case) must be decidable, i.e. there should exist a sound, complete and terminating algorithm deciding the problem; low computational complexity of the problem is an advantage.

- An ontology language should be expressive enough to cover the needs of ontology modellers (these were described in Section 1.2 for our setting).

- An ontology language should be empirically tractable, that is it should exhibit acceptable performance in real-world scenarios.
- Finally, an ontology language should be accessible to application experts, i.e. it should allow knowledge engineers to create ontological descriptions using intuitive constructs that are, e.g. closer to natural language rather than first-order logic notation.

As the requirements outlined above can be subjective and application-dependent, we choose to investigate them in the context of a real application which we consider to be representative of the various applications that focus on logic-based classification of structured objects and which was discussed in Section 1.2.

1.4 Thesis Outline

A synopsis of the remainder of this thesis is provided next:

Chapter 2 Provides preliminary definitions covering the syntax and stable model semantics of nonmonotonic existential rules, introduces the reasoning problems that we are interested in and recapitulates some computational complexity notions that are relevant to our work.

Chapter 3 Conducts an overview of existing logic-based formalisms for the representation of structured objects. Thereafter, it suggests nonmonotonic existential rules as an alternative framework for formally describing complex structures and identifies a number of difficulties that arise in that context, such as undecidability, existence of multiple stable models and the limited adoption of these rules by application experts.

Chapter 4 Presents a number of acyclicity conditions that ensure decidability for expressive fragments of existential rules with and without negation in the body; it also pinpoints the computational complexity for checking these conditions and for reasoning over the aforementioned fragments. It also analyses the relationship of

other acyclicity conditions originating from a variety of areas with our conditions and proves that our criteria strictly capture many of the existing ones.

Chapter 5 Introduces a novel stratification condition that guarantees stable model uniqueness for rule sets and properly generalises classical stratification. Tight complexity bounds are established for testing this condition and for the corresponding reasoning tasks. The chapter concludes with a discussion over a number of stratification conditions suggested in the 1990s and related work on nonmonotonic rules.

Chapter 6 Describes strengthenings of the previously introduced acyclicity and stratification conditions by exploiting positive constraints. Upper and lower complexity bounds are proved both for checking the extended conditions and for reasoning over the formalisms induced by the new conditions.

Chapter 7 Provides a high-level description of the sets of rules that were used for empirical evaluation; it also describes by means of examples the DGLP syntax—a 'surface syntax' for a fragment of nonmonotonic existential rules that was designed to make it easier for ontology developers to produce logical expressions for structured entities.

Chapter 8 Describes LoPStER, which is a DLV-based prototype that we developed for testing nonmonotonic existential rules in practice. The results of the experimental evaluation that involved classification of chemicals are presented and discussed. Previous efforts in the field of logic-based chemical classification are also reviewed and discussed.

Chapter 9 Summarises and highlights the significance of the results presented in this thesis. It concludes with suggestions for future work.

Appendix A Defines the DGLP syntax by a set of BNF derivation rules and their translation mappings to nonmonotonic existential rules.

Chapter 2

Foundational Definitions

The purpose of this chapter is to recapitulate some well-established definitions from the areas of logic programming and computational complexity. The formulation of these notions will serve as the basis for the knowledge representation formalisms presented in the following chapters as well as for the study of their properties.

Section 2.1 introduces the syntax of function-free Horn rules with negation in the body and existentially quantified variables in the head; Section 2.2 presents the stable model semantics; Section 2.3 describes the reasoning tasks which are of importance to the applications we describe and for which we are interested in providing suitable algorithms; finally, Section 2.4 reiterates a few computational complexity definitions that are relevant to our work.

2.1 Syntax of Nonmonotonic Existential Rules

In this section, we recall the definition of logic programming syntax for rules where non-monotonic negation is allowed in the body and skolemised existentially quantified variables in the head. For a detailed introduction and discussion of logic programming and answer set programming we refer the interested reader to the articles by Gelfond [91], Lifschitz [158] and Ferraris and Lifschitz [83].

Let $\mathcal{S} = (\mathbf{P}, \mathbf{V}, \mathbf{C}, \mathbf{F})$ be a *first-order logic signature* where $\mathbf{P}$ is a set of *predicate symbols*, $\mathbf{V}$ is a set of *variables*, $\mathbf{C}$ is a set of *constant symbols*, $\mathbf{F}$ is a set of *function*

symbols and $\mathbf{P}$, $\mathbf{V}$, $\mathbf{F}$ are countably infinite and pairwise disjoint sets and $\mathbf{C} \subseteq \mathbf{F}$. Let also $\mathsf{ar} : \mathbf{P} \cup \mathbf{F} \to \mathbb{N}$ be a mapping that associates each predicate and function symbol with a unique *arity* that is a non-negative integer. For each $\mathsf{f} \in \mathbf{F}$, if $\mathsf{ar}(\mathsf{f}) = 0$, then f is also in $\mathbf{C}$. For each predicate symbol Q such that $\mathsf{ar}(\mathsf{Q}) = n$ we say that Q is an *n-ary* predicate symbol (similarly for function symbols). For each first-order logic signature $\mathcal{S}$ and unless otherwise stated, we assume that $\mathbf{P}$ contains the nullary predicate $\bot$, which denotes a contradiction. The set of *terms* of a first-order logic signature $\mathcal{S}$ is the smallest set defined as follows:

1. Each variable and constant symbol is a term.

2. If $\mathsf{t}_1,\ldots,\mathsf{t}_n$ are terms and f is an n-ary function symbol such that $n \geq 1$, then $\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ is a term.

For each term t, the *depth* of t is a non-negative integer that is written as $\mathsf{depth}(\mathsf{t})$ and is defined as follows: if t is a variable or constant symbol, then $\mathsf{depth}(\mathsf{t}) = 0$; otherwise t is of the form $\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ and $\mathsf{depth}(\mathsf{t}) = 1 + \max_{i=1}^{n} \mathsf{depth}(\mathsf{t}_i)$. We abbreviate a list of items $\mathsf{t}_1,\ldots,\mathsf{t}_n$ with $\vec{\mathsf{t}}$ and define $|\vec{\mathsf{t}}| = n$; we treat lists as sets when order is irrelevant. A term t is a *subterm* of t' if t is the same as t' or if t' is of the form $\mathsf{f}(\vec{\mathsf{t}})$ and t is a subterm of some $\mathsf{t}'' \in \vec{\mathsf{t}}$. A term t_1 is a *proper subterm* of a term t_2 if t_1 is a *subterm* of t_2 and $\mathsf{depth}(\mathsf{t}_1) < \mathsf{depth}(\mathsf{t}_2)$. A term t is *ground* if t has no subterm which is a variable.

The set of *atoms* of a first-order logic signature $\mathcal{S}$ is the smallest set defined as follows:

1. If Q is a nullary predicate symbol, then Q is an atom.

2. If $\mathsf{t}_1,\ldots,\mathsf{t}_n$ are terms and Q is an n-ary predicate symbol with $n \geq 1$, then $\mathsf{Q}(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ is an atom.

A term t *occurs in* an atom α, if α is of the form $\mathsf{Q}(\vec{\mathsf{t}})$ and t is a subterm of some $\mathsf{t}' \in \vec{\mathsf{t}}$. An atom α is *ground* if each term that occurs in α is ground. A ground atom is also called a *fact*. In what follows, if the signature w.r.t. which a term (atom) is defined is clear from the context or irrelevant, then we simply refer to a term (atom).

For each atom α of the form $\mathsf{Q}(\vec{\mathsf{t}})$, we define $\mathsf{Pred}(\alpha) = \{\mathsf{Q}\}$; also, we define $\mathsf{Var}(\alpha)$, $\mathsf{Const}(\alpha)$ and $\mathsf{Terms}(\alpha)$ as the set of variables, constant symbols and terms that occur in α, respectively. For a set of atoms A, we set $\mathsf{Pred}(A) = \bigcup_{\alpha \in \mathsf{A}} \mathsf{Pred}(\alpha)$ and $\mathsf{Var}(\alpha)$, $\mathsf{Const}(\alpha)$ and $\mathsf{Terms}(\alpha)$ are defined analogously. An atom α is *function-free* if $\mathsf{Terms}(\alpha) = \mathsf{Var}(\alpha) \cup \mathsf{Const}(\alpha)$.

A *nonmonotonic existential rule* (or just *rule*) r is an expression of the form

$$\forall \vec{\mathsf{x}}.\forall \vec{\mathsf{z}}.[\beta_1 \wedge \ldots \wedge \beta_{\mathsf{m}} \wedge \mathbf{not}\ \beta_{\mathsf{m}+1} \wedge \ldots \wedge \mathbf{not}\ \beta_{\mathsf{n}} \rightarrow \exists \vec{\mathsf{y}}.\mathsf{h}_1 \wedge \ldots \wedge \mathsf{h}_\ell] \tag{2.1}$$

where $\beta_1, \ldots, \beta_{\mathsf{n}}, \mathsf{h}_1, \ldots, \mathsf{h}_\ell$ are function-free atoms, such that $\beta_1, \ldots, \beta_{\mathsf{n}}$, $\mathsf{m}, \mathsf{n} \geq 0$, $\ell > 0$ and $\vec{\mathsf{x}}$, $\vec{\mathsf{y}}$, $\vec{\mathsf{z}}$ are pairwise disjoint; also, $\mathsf{Var}(\vec{\beta}) = \vec{\mathsf{x}} \cup \vec{\mathsf{z}}$, $\mathsf{Var}(\vec{\mathsf{h}}) \subseteq \vec{\mathsf{x}} \cup \vec{\mathsf{y}}$ and $\vec{\mathsf{y}} \subseteq \mathsf{Var}(\vec{\mathsf{h}})$. In the remainder of this thesis, when we refer to a rule we imply a rule of the form (2.1) unless otherwise stated. We denote r with (B^+, B^-, H) where $B^+ = \{\beta_{\mathsf{i}}\}_{\mathsf{i}=1}^{\mathsf{m}}$, $B^- = \{\beta_{\mathsf{i}}\}_{\mathsf{i}=\mathsf{m}+1}^{\mathsf{n}}$ and $H = \{\mathsf{h}_{\mathsf{i}}\}_{\mathsf{i}=1}^{\ell}$. We refer to B^+ (B^-) as the *positive* (*negative*) body of the rule and to H as the *head* of the rule. In order to simplify the presentation we usually omit the universal quantification $\forall \vec{\mathsf{x}}.\forall \vec{\mathsf{z}}.$ in front of the rule; most of the time, we identify a conjunction of atoms with a set of atoms. We set $\mathsf{fr}(r) = \vec{\mathsf{x}}$ and we call $\mathsf{fr}(r)$ the *frontier* of r. If $\mathsf{m} = \mathsf{n}$, then r is a *positive* rule and if additionally $|\vec{\mathsf{y}}| = 0$, then r is a *datalog* rule; finally, if $\mathsf{H} = \{\bot\}$, then r is a *constraint*. For a rule $r = (\mathsf{B}^+, \mathsf{B}^-, \mathsf{H})$, we set $\mathsf{Pred}(r) = \mathsf{Pred}(\mathsf{B}^+) \cup \mathsf{Pred}(\mathsf{B}^-) \cup \mathsf{Pred}(\mathsf{H})$ and $\mathsf{Var}(r)$, $\mathsf{Const}(r)$ and $\mathsf{Terms}(r)$ are defined analogously. A *logic program* (or just *program*) P is a set of rules; a *positive* program is a set of positive rules and a *datalog* program is a set of datalog rules.

A *substitution* σ is a mapping $\theta : \mathbf{V} \rightarrow \mathbf{T}$. Application of a substitution σ is recursively defined as follows; for a variable v we set $\mathsf{v}\sigma = \sigma(\mathsf{v})$; for a constant symbol c we set $\mathsf{c}\sigma = \mathsf{c}$ and for a function term $\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_{\mathsf{n}})$ we set $[\mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_{\mathsf{n}})]\sigma = \mathsf{f}(\mathsf{t}_1\sigma, \ldots, \mathsf{t}_{\mathsf{n}}\sigma)$; for a vector $\vec{\mathsf{t}}$ consisting of n terms we set $\vec{\mathsf{t}}\sigma = \mathsf{t}_1\sigma, \ldots, \mathsf{t}_{\mathsf{n}}\sigma$; moreover, for an atom $\mathsf{Q}(\vec{\mathsf{t}})$ we set $[\mathsf{Q}(\vec{\mathsf{t}})]\sigma = Q(\vec{\mathsf{t}}\sigma)$ and for a set of atoms A we set $A\sigma = \bigcup_{\alpha \in A}\{\alpha\sigma\}$. For two sets of atoms A and A', a *homomorphism* from A to A' is a substitution $\theta : \mathsf{Var}(A) \rightarrow \mathsf{Terms}(A')$, such that $A\theta \subseteq A'$. For each set of rules P we assume w.l.o.g. that for each $r, r' \in P$ different from each other we have $\mathsf{Var}(r) \cap \mathsf{Var}(r') = \emptyset$.

Let r be a rule of the form (2.1); the *skolemisation* $\mathsf{sk}(r)$ of r is the rule

$$\forall \vec{\mathsf{x}}.\forall \vec{\mathsf{z}}.[\beta_1 \wedge \ldots \wedge \beta_{\mathsf{m}} \wedge \mathbf{not}\ \beta_{\mathsf{m}+1} \wedge \ldots \wedge \mathbf{not}\ \beta_{\mathsf{n}} \rightarrow (\mathsf{h}_1 \wedge \ldots \wedge \mathsf{h}_\ell)\theta] \tag{2.2}$$

where $\theta = \{\mathsf{y} \mapsto \mathsf{f}_{\mathsf{y}}(\vec{\mathsf{x}})) \mid \mathsf{y} \in \vec{\mathsf{y}}\}$ such that f_{y} is an $|\vec{\mathsf{x}}|$-ary function symbol unique for each $\mathsf{y} \in \vec{\mathsf{y}}$; we say that $\mathsf{sk}(r)$ is a *skolemised* rule. For a logic program P, we set $\mathsf{sk}(P) = \{\mathsf{sk}(r) \mid r \in P\}$; a logic program P is *skolemised* if every rule in P is skolemised.

Example 1. Let P be a program that consists of the rules r_1–r_6 and the fact $\mathsf{mercuryChloride}(\mathsf{m})$. Rule r_1 describes the structure of mercury chloride, which is a toxic chemical compound, in terms of chemical elements of its atoms and chemical bonds. Rules r_2 and r_3 define the chemical classes of organic and inorganic molecules, whereas r_4 and r_5 model knowledge about toxicity.

$$r_1 : \quad \mathsf{mercuryChloride}(\mathsf{x}) \rightarrow \exists \mathsf{y}_1, \mathsf{y}_2, \mathsf{y}_3. \bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom}(\mathsf{x}, \mathsf{y}_\mathsf{i}) \wedge \mathsf{molecule}(\mathsf{x}) \wedge \mathsf{chlorine}(\mathsf{y}_1) \wedge \mathsf{mercury}(\mathsf{y}_2) \wedge \mathsf{chlorine}(\mathsf{y}_3) \wedge \mathsf{singleBond}(\mathsf{y}_1, \mathsf{y}_2) \wedge \mathsf{singleBond}(\mathsf{y}_2, \mathsf{y}_3)$$

$$r_2 : \quad \mathsf{singleBond}(\mathsf{x}_1, \mathsf{x}_2) \rightarrow \mathsf{singleBond}(\mathsf{x}_2, \mathsf{x}_1)$$

$$r_3 : \quad \mathsf{hasAtom}(\mathsf{x}_3, \mathsf{z}_3) \wedge \mathsf{carbon}(\mathsf{z}_3) \rightarrow \mathsf{organic}(\mathsf{x}_3)$$

$$r_4 : \quad \mathsf{molecule}(\mathsf{x}_4) \wedge \mathbf{not}\ \mathsf{organic}(\mathsf{x}_4) \rightarrow \mathsf{inorganic}(\mathsf{x}_4)$$

$$r_5 : \quad \mathsf{mercuryChloride}(\mathsf{x}_5) \rightarrow \mathsf{highlyToxic}(\mathsf{x}_5)$$

$$r_6 : \quad \mathsf{highlyToxic}(\mathsf{x}_6) \rightarrow \exists \mathsf{y}_6.\mathsf{hasProperty}(\mathsf{x}_6, \mathsf{y}_6) \wedge \mathsf{highToxicity}(\mathsf{y}_6)$$

Note that r_1, r_2, r_3, r_5 and r_6 are positive rules and r_2, r_3 and r_5 are datalog rules. The skolemisations of r_1 and r_6 are as follows.

$$\mathsf{sk}(r_1) : \mathsf{mercuryChloride}(\mathsf{x}) \rightarrow \bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom}(\mathsf{x}, \mathsf{f}_\mathsf{i}(\mathsf{x})) \wedge \mathsf{molecule}(\mathsf{x}) \wedge \mathsf{chlorine}(\mathsf{f}_1(\mathsf{x})) \wedge \mathsf{mercury}(\mathsf{f}_2(\mathsf{x})) \wedge \mathsf{chlorine}(\mathsf{f}_3(\mathsf{x})) \wedge \mathsf{singleBond}(\mathsf{f}_1(\mathsf{x}), \mathsf{f}_2(\mathsf{x})) \wedge \mathsf{singleBond}(\mathsf{f}_2(\mathsf{x}), \mathsf{f}_3(\mathsf{x}))$$

$$\mathsf{sk}(r_6) : \quad \mathsf{highlyToxic}(\mathsf{x}_6) \rightarrow \mathsf{hasProperty}(\mathsf{x}_6, \mathsf{f}_6(\mathsf{x}_6)) \wedge \mathsf{highToxicity}(\mathsf{f}_6(\mathsf{x}_6))$$

The skolemisation of P is $\mathsf{sk}(P) = \{\mathsf{sk}(r_1), r_2, r_3, r_4, r_5, \mathsf{sk}(r_6)\}$. Please note that the rules provided here are *guarded*, that is each rule has a body atom where all

universally quantified variables occur. However, as we will see later in this thesis (e.g. Example 5 on page 50) non-guarded rules are also essential for the modelling of the domains we are interested in. ♢

2.2 Stable Model Semantics

We next introduce the *stable model semantics*, which is one of the most widely adopted semantics for logic programming rules. For a detailed presentation and extensive discussion of stable model semantics, one can consult the article where stable model semantics were defined for the first time [92].

Let P be a skolemised logic program. The *Herbrand Universe* of P (written $\mathsf{HU}(P)$) is the set of all terms that can be formed using $\mathsf{Const}(P)$ and the function symbols of P; if $\mathsf{Const}(P)$ is empty, then an arbitrary constant symbol c_0 is used. The *Herbrand Base* of P (written $\mathsf{HB}(P)$) is the set of all ground atoms constructed using the predicates in $\mathsf{Pred}(P)$ and the terms in $\mathsf{HU}(P)$. The grounding of a rule r w.r.t. a set of terms T is the set of rules obtained by uniformly substituting the variables of r with the terms of T in all possible ways. The program $\mathsf{ground}(P)$ is obtained from P by replacing each rule $r \in P$ with its grounding w.r.t. $\mathsf{HU}(P)$.

Let $I \subseteq \mathsf{HB}(P)$ be a set of ground atoms. Then, I *satisfies* a ground rule $r = (B^+, B^-, H)$ (written $I \models r$) if $B^+ \subseteq I$ and $B^- \cap I = \emptyset$ imply $H \subseteq I$. We say that I is a *model* of a program P, written $I \models P$, if $\bot \notin I$ and I satisfies each rule $r \in \mathsf{ground}(P)$. The set I is a *minimal model* of P if $I \models P$ and no $I' \subsetneq I$ exists such that $I' \models P$. The *Gelfond-Lifschitz reduct* $\mathsf{GL}(P, I)$ of a logic program P w.r.t. I is obtained from $\mathsf{ground}(P)$ by removing each rule $(B^+, B^-, H) \in \mathsf{ground}(P)$ such that $B^- \cap I \neq \emptyset$, and removing all atoms **not** β_i in all the remaining rules. A set I is a *stable model* of P (written $I \models_{\mathsf{SM}} P$) if I is a minimal model of $\mathsf{GL}(P, I)$. Please note that a program P can have more than one stable models. Given a fact α, we write $P \models \alpha$ if $\alpha \in I$ for each stable model I of P; otherwise, we write $P \not\models \alpha$.

Example 2. For the program P given in Example 1, the only stable model of $\mathsf{sk}(P) \cup \mathsf{mercuryChloride}(\mathsf{m})$ is the following set $\mathcal{M}$.

$$\begin{aligned}\mathcal{M} = \{&\mathsf{mercuryChloride(m)}, \mathsf{molecule(m)}, \mathsf{hasAtom(m, f_1(m))}, \mathsf{hasAtom(m, f_2(m))},\\ &\mathsf{hasAtom(m, f_3(m))}, \mathsf{chlorine(f_1(m))}, \mathsf{mercury(f_2(m))}, \mathsf{chlorine(f_3(m))}\\ &\mathsf{singleBond(f_1(m), f_2(m))}, \mathsf{singleBond(f_2(m), f_1(m))}, \mathsf{singleBond(f_2(m), f_3(m))},\\ &\mathsf{singleBond(f_3(m), f_2(m))}, \mathsf{inorganic(m)}, \mathsf{highlyToxic(m)}, \mathsf{hasProperty(m, f_5(m))},\\ &\mathsf{highToxicity(f_5(m))}\}\end{aligned}$$

In order to verify that $\mathcal{M}$ is the stable model of $\mathsf{sk}(P) \cup \{\mathsf{mercuryChloride(m)}\}$, let us consider $P' = \{r_i\}_{i=6}^{11}$ which is the subset of $\mathsf{ground}(\mathsf{sk}(P) \cup \{\mathsf{mercuryChloride(m)}\})$ such that P' is not trivially satisfied by $\mathcal{M}$—that is for each rule $(B^+, B^-, H) \in P'$ it holds that $B^+ \subseteq \mathcal{M}$ and $B^- \cap \mathcal{M} = \emptyset$.

$$\begin{aligned}
r_6 :&\quad \mathsf{mercuryChloride(m)} \rightarrow \bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom(m, f_i(m))} \wedge \mathsf{molecule(m)} \wedge\\
&\qquad \mathsf{chlorine(f_1(m))} \wedge \mathsf{mercury(f_2(m))}) \wedge\\
&\qquad \mathsf{chlorine(f_3(m))} \wedge \mathsf{singleBond(f_1(m), f_2(m))} \wedge\\
&\qquad \mathsf{singleBond(f_2(m), f_3(m))}\\
r_7 :&\quad \mathsf{singleBond(f_1(m), f_2(m))} \rightarrow \mathsf{singleBond(f_2(m), f_1(m))}\\
r_8 :&\quad \mathsf{singleBond(f_2(m), f_3(m))} \rightarrow \mathsf{singleBond(f_3(m), f_2(m))}\\
r_9 :&\quad \mathsf{singleBond(f_2(m), f_1(m))} \rightarrow \mathsf{singleBond(f_1(m), f_2(m))}\\
r_{10} :&\quad \mathsf{singleBond(f_3(m), f_2(m))} \rightarrow \mathsf{singleBond(f_2(m), f_3(m))}\\
r_{11} :&\quad \mathsf{molecule(m)} \wedge \mathbf{not}\ \mathsf{organic(m)} \rightarrow \mathsf{inorganic(m)}\\
r_{12} :&\quad \mathsf{mercuryChloride(m)} \rightarrow \mathsf{highlyToxic(m)}\\
r_{13} :&\quad \mathsf{highlyToxic(m)} \rightarrow \mathsf{hasProperty(m, f_5(m))} \wedge \mathsf{highToxicity(f_5(m))}
\end{aligned}$$

The GL-reduct of P' w.r.t. $\mathcal{M}$ is $\mathsf{GL}(P', \mathcal{M}) = \{r_6, r_7, r_8, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}\}$, where r_{14} is defined as follows.

$$r_{14} : \mathsf{molecule(m)} \rightarrow \mathsf{inorganic(m)}$$

$\mathcal{M}$ is clearly the minimal model of $\mathsf{GL}(P', \mathcal{M})$. ♢

2.3 Reasoning Tasks

In this section, we describe the reasoning tasks that we will be studying in the subsequent chapters. All reasoning tasks that we define next can be directly reduced to *fact entailment*, i.e. for a program P and a fact α deciding whether $\mathsf{sk}(P) \models \alpha$ is true; fact entailment for a program P can be decided by computing all stable models of $\mathsf{sk}(P)$ and checking if α is in each such model.

Let P be a program, let A and B be unary predicates such that $\mathsf{A}, \mathsf{B} \in \mathsf{Pred}(P)$ and let c be a constant such that $\mathsf{c} \notin \mathsf{Const}(\mathsf{sk}(P))$. We say that A *is subsumed by* B w.r.t. P (written $P \models \mathsf{A} \sqsubseteq \mathsf{B}$), if $\mathsf{sk}(P) \cup \{\mathsf{A}(\mathsf{c})\} \models \mathsf{B}(\mathsf{c})$. The problem of *subsumption entailment* for P and A, B is to determine whether $P \models \mathsf{A} \sqsubseteq \mathsf{B}$. The problem of *subsumer computation* for P and A consists in retrieving the set S of unary predicates such that $S \subseteq \mathsf{Pred}(P)$ and $P \models \mathsf{A} \sqsubseteq B$ for each $\mathsf{B} \in S$.

Example 3. For the program P of Example 1, it is $P \models \mathsf{mercuryChloride} \sqsubseteq \mathsf{inorganic}$ and $P \models \mathsf{mercuryChloride} \sqsubseteq \mathsf{highlyToxic}$. These two subsumptions are a consequence of the entailment $\mathsf{sk}(P) \cup \{\mathsf{mercuryChloride}(\mathsf{m})\} \models \mathsf{inorganic}(\mathsf{m})$ and the entailment $\mathsf{sk}(P) \cup \{\mathsf{mercuryChloride}(\mathsf{m})\} \models \mathsf{highlyToxic}(\mathsf{m})\}$ which in turn hold by the inclusion $\{\mathsf{inorganic}(\mathsf{m}), \mathsf{highlyToxic}(\mathsf{m})\} \subseteq \mathcal{M}$ and the fact that $\mathcal{M}$ is the only stable model of $\mathsf{sk}(P) \cup \{\mathsf{mercuryChloride}(\mathsf{m})\}$ (uniqueness of $\mathcal{M}$ follows from the fact that P is stratified—stratification is a notion that will be introduced in Section 3.3.2). ♢

2.4 Computational Complexity

We assume some core computational complexity notions whose formal definition is beyond the scope of this thesis, such as decision problems, decidability, complexity classes, (non-)deterministic Turing machines (often abbreviated with (N)DTM) and oracle calls. Comprehensive introductions to the theory of computational complexity can be found in the textbooks by Garey and Johnson [88] and Papadimitriou [194]. We also assume the definition of the complexity classes that appear in Figure 2.1 and which are the classes considered in this thesis.

For the classes P, ExpTime and 2ExpTime, it follows from the time hierarchy theorem [109] that P $\subsetneq$ ExpTime $\subsetneq$ 2ExpTime. The subclass relations P $\subseteq$ NP and P $\subseteq$ coNP also hold; although it is not known whether they are strict, most experts believe that both inclusions are proper [115]. Moreover, the inclusions NP $\subseteq \Delta_2^P$, coNP $\subseteq \Delta_2^P$, $\Delta_2^P \subseteq \Sigma_2^P$ and $\Delta_2^P \subseteq \Pi_2^P$ hold, but it is not known whether they are strict; if they are not, then NP = coNP = $\Delta_2^P = \Sigma_2^P = \Pi_2^P$. Figure 2.1 also illustrates the hierarchy among the discussed complexity classes; a pair of lines with at least one interrupted line denotes a superclass relation that is not known to be proper; a pair of continuous lines is used when the inclusion has been proved to be strict.

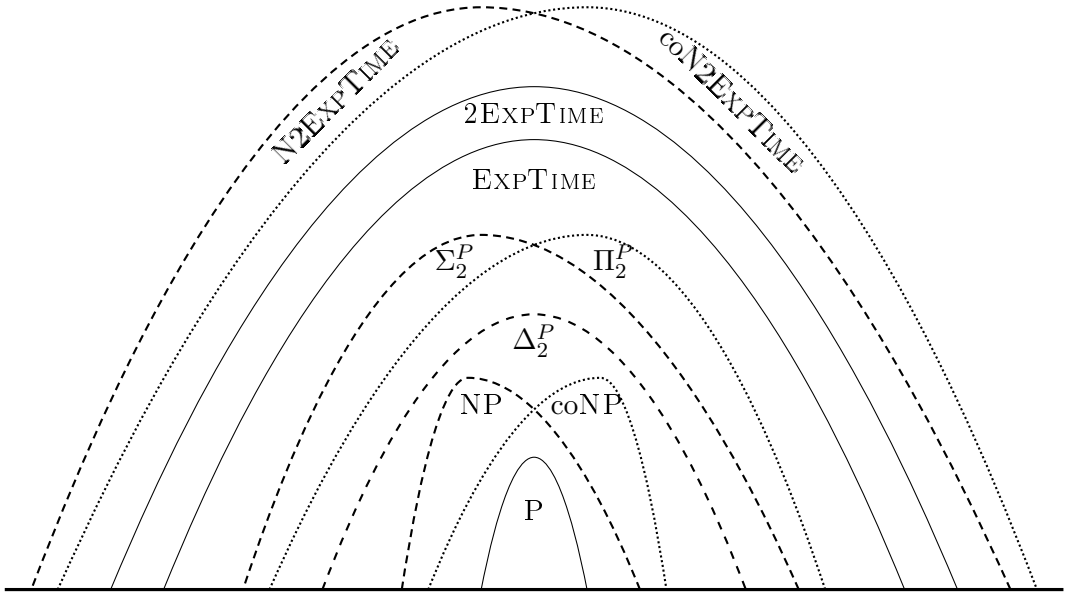


Figure 2.1: Complexity classes considered in this thesis, where a pair of lines with at least one interrupted line denotes an inclusion that is not known to be proper and a pair of continuous lines denotes a proper inclusion

In the context of logic programming languages and in particular for the decision problems where part of the decision problem instance is a logic program P and a set of facts F, the focus is on three main kinds of complexity [232]:

- The *data complexity* which is measured in the number of facts, that is the size of F.
- The *program complexity* which is measured in the number of rules with non-

empty body, that is the size of P.

- The *combined complexity* which is measured in the number of both facts and rules, that is the size of $P \cup F$.

Chapter 3

Modelling Structured Objects: State of the Art

The scope of this chapter is to present the state of the art w.r.t. available logic-based languages for the modelling of structured entities. Furthermore, it suggests a new approach for the representation of complex structures and it highlights some open problems that arise from this suggestion; it also investigates a number of solutions to these problems.

Section 3.1 surveys various ontology languages that can be used for the description of structured objects and points out the main strengths and shortcomings of each language by means of a running example; Section 3.2 presents a new proposal for the representation of structured objects by introducing modelling based on nonmonotonic existential rules; lastly, Section 3.3 discusses the difficulties that emerge in the context of modelling and reasoning with nonmonotonic existential rules and enumerates a number of solutions that have been proposed to tackle them.

3.1 Survey of Available KR Formalisms

The current section offers an overview of logic-based formalisms for the representation of structured objects. In most cases, the syntax and semantics for a decidable fragment of the language is presented; this is followed by an assessment of the expressivity

of the said languages for the purposes of our modelling by means of a realistic use case. Our modelling example is taken from the chemical domain and is first described after introducing the syntax and semantics of OWL.

3.1.1 OWL

The Web Ontology Language (OWL) [130] and its latest version OWL 2 [63] are families of W3C standardised languages that have played a pivotal role in the advent of Semantic Technologies—a set of datacentric techniques and algorithms that seek to derive meaning from information while processing it. Due to the availability of highly optimised OWL reasoners [136, 137, 231, 95, 219], OWL has been extensively used in the deployment of numerous knowledge management systems spanning from proof-of-concept ontologies to large-scale knowledge bases.

As we saw earlier, Description logics (DL) [12] constitute a family of logic-based formalisms with robust computational properties that form the theoretical foundation of the OWL family of languages. The DL syntax defines axioms with the help of concepts, roles and individuals. Individuals are similar to constants, that is names for elements of the domain; concepts (unary predicates) describe common characteristics shared by a set of individuals; finally, roles (binary predicates) denote links between pairs of individuals. The DL semantics is based on first-order logic (FOL) and as such does not impose any minimality condition on the models and adopts the open-world assumption.

We formally introduce the syntax of $\mathcal{SROIQ}$ [127], which is the description logic language underpinning OWL 2 DL, the latest most expressive and decidable variant of OWL. Let $\mathcal{S}_{\mathsf{DL}} = (N_C, N_R, N_I)$ be a *description logic signature* where N_C is a set of *concept names*, N_R is a set of *role names* and N_I is a set of *individual names*, such that N_C, N_R and N_I are countably infinite and pairwise disjoint.

The set of roles is the set $N_R \cup \{R^- \mid R \in N_R\} \cup \{U\}$, where $U \notin N_R$ is a special role that we call the *universal role*. A *role axiom* α is defined as indicated by Table 3.1. A *regular order* $\prec$ is an irreflexive transitive relation on the set of roles such that $S \prec R$ if $S^- \prec R$ where R and S are roles. Let $\prec$ be a regular order. A

Name	Role axiom α	Condition for $\mathcal{I} \models \alpha$
Role inclusion	$R_1 \circ \ldots \circ R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
Disjointness	$\mathsf{Dis}(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

Table 3.1: Syntax and semantic of role axioms in $\mathcal{SROIQ}$

role inclusion axiom (RIA) of the form $R_1 \circ \ldots \circ R_n \sqsubseteq R$ is $\prec$-regular if R is a role name and

1. $n = 2$ and $R_1 = R_2 = R$ or
2. $n = 1$ and $R_1 = R^-$ or
3. $R_i \prec R$ for $1 \leq i \leq n$ or
4. $R_1 = R$ and $R_i \prec R$ for $2 \leq i \leq n$ or
5. $R_n = R$ and $R_i \prec R$ for $1 \leq i \leq n - 1$.

A *role hierarchy* is a finite set of RIAs. A *regular* role hierarchy $\mathcal{R}_h$ is a role hierarchy if there exists a regular order $\prec$ such that each RIA in $\mathcal{R}_h$ is $\prec$-regular. For each role hierarchy $\mathcal{R}_h$, the set of *non-simple* roles is given by the following rules [150]:

1. The universal role U is non-simple.
2. If $\mathcal{R}_h$ contains an axiom of the form $R_1 \circ \ldots \circ R_n \sqsubseteq R$, then R is non-simple.
3. For each $R \in N_R$, if R is non-simple, then R^- is also non-simple (if R is of the form S^-, then R^- should be read as S).
4. For each $R, S \in N_R$, if R is non-simple and $\mathcal{R}_h$ contains an axiom of the form $R \sqsubseteq S$, then S is non-simple.

All other roles are called simple.

For each set of disjointness axioms $\mathcal{R}_d$, $\mathcal{R}_d$ is *simple* if each role appearing in $\mathcal{R}_d$ is simple. An *RBox* is a set $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_d$, where $\mathcal{R}_h$ is a regular role hierarchy and $\mathcal{R}_\alpha$ is a finite, simple set of disjointness axioms.

Given a (possibly empty) RBox $\mathcal{R}$, the set of concepts is recursively defined using the constructors listed in the middle column of Table 3.2, where $A \in N_C$, C and D are concepts, R is a role, S is a simple role, $a \in N_I$ and n is a non-negative integer. A *concept inclusion axiom* is an expression of the form $C \sqsubseteq D$, where C and D are concepts. A *TBox* $\mathcal{T}$ is a finite set of concept inclusion axioms.

Name	Syntax	Semantics
Concept name	A	$A^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} :\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Self concept	$\exists S.\mathsf{Self}$	$\{x \in \Delta^{\mathcal{I}} \mid (x,x) \in S^{\mathcal{I}}\}$
Universal	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} :\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
At least	$(\geq n)S.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} :\ (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$
At most	$(\leq n)S.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} :\ (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$
Exactly	$(= n)S.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} :\ (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} = n\}$

Table 3.2: Syntax and semantics of concepts in $\mathcal{SROIQ}$

An *assertion* is an expression of the form $R(a,b)$, $\neg R(a,b)$, $C(a)$ and $a \not\approx b$, where $a, b \in N_I$, R is a role and C is a concept. An *ABox* $\mathcal{A}$ is a finite set of assertions. An *axiom* is a role axiom, a concept inclusion axiom or an assertion. A $\mathcal{SROIQ}$ *knowledge base* is a triple of the form $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, where $\mathcal{R}$, $\mathcal{T}$, $\mathcal{A}$ is an RBox, TBox and ABox, respectively.

An interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set which we call the *domain* of $\mathcal{I}$ and $\cdot^{\mathcal{I}}$ is a mapping which we call the *valuation*. The valuation maps each role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that $(R^-)^{\mathcal{I}} = \{(b,a) \mid (a,b) \in R^{\mathcal{I}}\}$ and $U^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, each $C \in N_C$ to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each $i \in N_I$ to a member of

$\Delta^{\mathcal{I}}$. The concepts are interpreted as indicated in the righthand column of Table 3.2. For a role axiom α, $\mathcal{I} \models \alpha$ if the corresponding condition of Table 3.1 holds, where $\circ$ when applied to binary relations denotes composition thereof; for a concept inclusion axiom $C \sqsubseteq D$, $\mathcal{I} \models C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; we have $\mathcal{I} \models R(a,b)$, $\mathcal{I} \models \neg R(a,b)$, $\mathcal{I} \models \neg C(a)$ and $\mathcal{I} \models a \not\approx b$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$, respectively. For an axiom α, we say that $\mathcal{I}$ *satisfies* α if $\mathcal{I} \models \alpha$. For a $\mathcal{SROIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, we say that $\mathcal{I}$ is a model of $\mathcal{K}$ (written $\mathcal{I} \models \mathcal{K}$) if $\mathcal{I} \models \alpha$ for each $\alpha \in \mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$; RBox, TBox and ABox models are defined analogously. For a $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$ and an axiom α, we say that $\mathcal{K}$ entails α (written $\mathcal{K} \models \alpha$) if for every model $\mathcal{I}$ of $\mathcal{K}$, we have $\mathcal{I} \models \alpha$; entailment of an axiom by an RBox, TBox or ABox is defined analogously.

Some common inference problems w.r.t. a knowledge base $\mathcal{K}$ are:

1. Checking whether a knowledge base $\mathcal{K}$ is consistent, that is whether $\mathcal{K}$ has a model (*knowledge base consistency*).

2. Checking whether a concept A is satisfiable, that is whether $A^{\mathcal{I}}$ is non-empty for some model $\mathcal{I}$ of $\mathcal{K}$ (*concept satisfiability*).

3. Checking whether a concept A is subsumed by a concept B w.r.t. a knowledge base $\mathcal{K}$ ($\mathcal{K} \models A \sqsubseteq B$), that is if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds for every model $\mathcal{I}$ of the knowledge base $\mathcal{K}$ (*subsumption*).

The problem of concept satisfiability for a $\mathcal{SROIQ}$ knowledge has been proved to be decidable [127] and N2ExpTime-complete [135]; given that knowledge base consistency and subsumption are polynomially reducible to concept unsatisfiability [14], it is coN2ExpTime-complete to decide them.

Before embarking on a discussion about the expressive power of OWL we introduce the chemical domain example that we will use throughout the rest of this chapter. Our modelling scenario is that of an ontology engineer who wishes to formally describe the molecules of benzene and cyclobutane whose arrangement of atoms and bonds is depicted in Figures 3.1(a) and 3.1(b), respectively. Furthermore, the engineer would like to create ontological descriptions for the chemical classes of

(i) hydrocarbons, that is carbon-containing molecules that consist exclusively of hydrogen or carbon,

(ii) molecules with a six-membered ring, that is molecules that contain a cyclic arrangement of six atoms and

(iii) molecules with a benzene ring, that is molecules that contain a six-membered ring of carbon atoms with alternating single and double bonds.

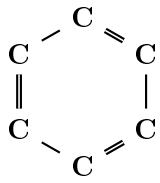
Finally, the engineer wants to use an automated reasoning tool that will infer that benzene and cyclobutane are hydrocarbons, that benzene is a molecule with a six-membered ring and a benzene ring, and nothing else.

Let us now have a closer look at the expressive power of OWL regarding the representation and classification of structured objects. We identify two main limitations.

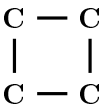
The first limitation arises from the forest-likeness that characterises the models of Description Logics ontologies. Assume that our knowledge engineer wishes to describe with the help of OWL the benzene and cyclobutane molecules. We describe the modelling for benzene; the case of cyclobutane is similar. One can represent benzene as a molecule that has at least six carbon atoms each of which has exactly one single bond with a carbon atom and exactly one double bond with another carbon atom. Axiom α_1 encodes this description in DL.

$$\mathsf{Benzene} \sqsubseteq \mathsf{Molecule} \sqcap (\geq 6)\mathsf{hasAtom}.[\mathsf{Carbon} \sqcap (=1)\mathsf{singleBond}.\mathsf{Carbon} \sqcap (=1)\mathsf{doubleBond}.\mathsf{Carbon}] \qquad (\alpha_1)$$

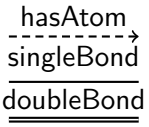
Let $\mathcal{R}_1 = \{\mathsf{singleBond} \sqsubseteq \mathsf{singleBond}^-, \mathsf{doubleBond} \sqsubseteq \mathsf{doubleBond}^-\}$, that is let singleBond and doubleBond be symmetric roles, and let $\mathcal{K}_1 = (\mathcal{R}_1, \{\alpha_1\}, \emptyset)$. Figures 3.1(d) and 3.1(e) illustrate two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, where the squares correspond to distinct objects of the interpretation domain, the set of squares labelled with a concept C is the interpretation of C and the set of edges labelled with a role R is the interpretation of R (the label assigned to each edge is provided by the key and arrowless edges are assumed to be bidirectional). One can readily check that both interpretations satisfy $\mathcal{K}_1$. However, only interpretation $\mathcal{I}_1$ is the *canon-*



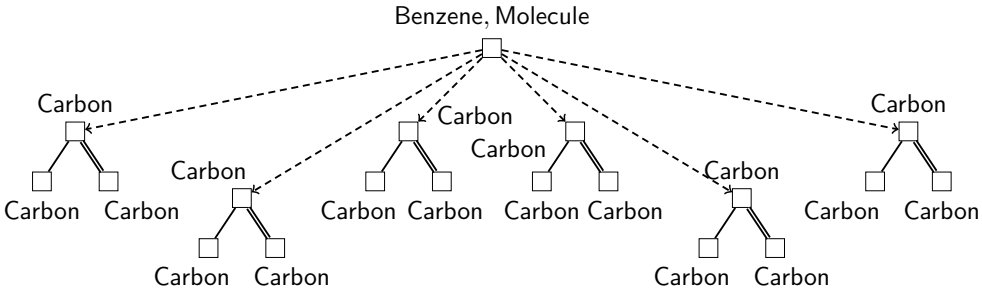
(a) Benzene chemical graph



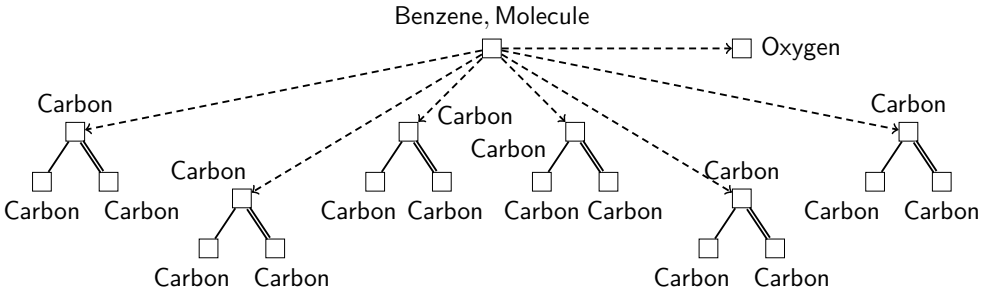
(b) Cyclobutane chemical graph



(c) Key



(d) Interpretation $\mathcal{I}_1$ (OWL canonical model)



(e) Interpretation $\mathcal{I}_2$ (OWL additional model)

Figure 3.1: Chemical structures and OWL models of benzene molecule

ical model of $\mathcal{K}_1$, that is the model that will be constructed by standard tableaux algorithms to perform logical inference. Nevertheless, interpretation $\mathcal{I}_1$ does not reflect the cyclic structure of benzene, since $\mathcal{I}_1$ captures the fact that benzene contains six carbon atoms but not that they are arranged in a ring. In fact, it is *not possible* to faithfully describe cyclic structures using a DL-based language because of a variant of the *forest-model property* [233, 98] that is exhibited by Description Logics. The forest-model property, to which favourable computational properties of Description Logics such as decidability are attributed, obliges each satisfiable DL knowledge base to have at least one forest-like model. Additionally, since the DL syntax does not offer access to variables, one is not able, e.g., to recognise structures that contain a six-membered ring (that is a ring of length six) such as benzene. As

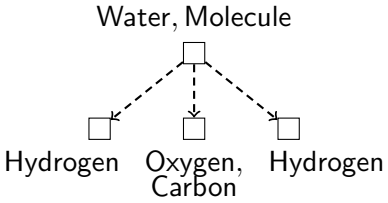


Figure 3.2: Interpretation $\mathcal{I}_3$ (OWL additional model)

a consequence, one is not able provide a suitable definition for a concept with, e.g., the name MoleculeWithSixMemberedRing and then use a DL-based reasoner to derive Benzene $\sqsubseteq$ MoleculeWithSixMemberedRing.

The second expressivity shortcoming results from the first-order logic semantics of Description Logics that do not impose any minimality condition on the size of the models of a DL knowledge base. Suppose that a modeller wants to represent the class of hydrocarbon molecules; that is, all the compounds that consist entirely of carbon or hydrogen atoms. One may use the universal concept constructor for this purpose, that is extend $\mathcal{K}_1$ with axiom α_2.

$$\mathsf{Molecule} \sqcap \forall \mathsf{hasAtom}.(\mathsf{Hydrogen} \sqcup \mathsf{Carbon}) \sqsubseteq \mathsf{Hydrocarbon} \qquad (\alpha_2)$$

Let also the following axiom be used to identify all molecules that contain an oxygen atom

$$\mathsf{Molecule} \sqcap \exists \mathsf{hasAtom}.\mathsf{Oxygen} \sqsubseteq \mathsf{OxygenMolecule} \qquad (\alpha_3)$$

Now, let $\mathcal{K}_2 = (\mathcal{R}_1, \{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$. In order for $\mathcal{K}_2$ to entail that benzene is a subclass of hydrocarbon, for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}_2$ we need to have $\mathcal{I} \models \mathsf{Benzene} \sqsubseteq \mathsf{Hydrocarbon}$. However, there exists at least one model of $\mathcal{K}_2$, namely interpretation $\mathcal{I}_2$ from Figure 3.1(e), that does not satisfy the said subsumption, that is

$$\mathcal{I}_2 \models \mathcal{K}_2 \qquad (3.1)$$

$$\mathcal{I}_2 \not\models \mathsf{Benzene} \sqsubseteq \mathsf{Hydrocarbon} \qquad (3.2)$$

This is due to the open-world assumption (OWA) adopted by Description Logic semantics, according to which missing information is treated as *not known* rather than

false. Treating non-provable statements as not holding corresponds to the closed-world assumption (CWA), also known as *negation-as-failure*, where failing to entail a statement implies the falsehood of the statement. In the context of the benzene example, there is no evidence as to whether benzene *does not* contain an oxygen and, so, the interpretation where it also contains an oxygen is still a model of $\mathcal{K}_2$. Consequently, an OWL reasoner would fail to classify benzene as a hydrocarbon for the given knowledge base, that is

$$\mathcal{K}_2 \not\models \mathsf{Benzene} \sqsubseteq \mathsf{Hydrocarbon} \tag{3.3}$$

Note that one could turn (3.3) into an entailment by changing the cardinality restriction of axiom α_1 from 'at least 6' to 'exactly 6'. However, a similar problem that is less easy to repair arises, when e.g. one wants to classify water as an inorganic molecule, that is a molecule that does not contain carbon. One can model the water molecule using axiom (3.4) and try to recognise it as inorganic using axiom (3.5).

$$\mathsf{Water} \sqsubseteq \mathsf{Molecule} \sqcap (=1)\mathsf{hasAtom.Oxygen} \sqcap (=2)\mathsf{hasAtom.Hydrogen} \tag{3.4}$$

$$\mathsf{Molecule} \sqcap \forall \mathsf{hasAtom.}(\neg \mathsf{Carbon}) \sqsubseteq \mathsf{Inorganic} \tag{3.5}$$

In that case, the additional interpretation $\mathcal{I}_3$ from Figure 3.2 still satisfies axiom 3.4, which prevents water from being recognised as inorganic. In order to derive this subsumption, one can insert a disjointness axiom such as $\mathsf{Carbon} \sqcap \mathsf{Oxygen} \sqsubseteq \bot$ that eliminates model $\mathcal{I}_3$. However, in such a scenario one would need to add similar disjointness pairs for all chemical elements. Given that there are as many as 118 elements, this is clearly a cumbersome solution that may even be infeasible when the parts of a complex object are labelled with elements of a much broader domain, such as genes.

The failure of the OWL knowledge bases above to entail the subsumptions without extra axioms does not imply that FOL-based semantics are inherently ill-suited for formal representation of knowledge. The design of a semantics for an ontology language—and by extension the choice between open- and closed-world assumption—should be driven by the requirements of the target domain. For example, opting for a

first-order logic semantics in a web setting where the available descriptions of objects are usually incomplete is a well-justified decision. At the same time, for domains such as life sciences where most often the entire structure of entities is known, CWA-based semantics may be a reasonable option because it enables the performance of tasks, such as structure-dependent classification, that are difficult to achieve with FOL semantics.

An alternative approach to choosing between open and closed world assumption is to adopt closed world assumption only partially by 'locally closing' a role with the help of the autoepistemic operator **K** [104]. The use of **K**-operator in front of a role r changes the semantics so that $\mathbf{K}r$ is interpreted as the intersection of role extensions over all first-order interpretations. For example, we could exploit this construct to define hydrocarbon as

$$\mathsf{Molecule} \sqcap \forall \mathbf{K}\mathsf{hasAtom}.(\mathsf{Hydrogen} \sqcup \mathsf{Carbon}) \sqsubseteq \mathsf{Hydrocarbon}$$

and obtain that benzene is a subclass of hydrocarbon. However, this approach would still be problematic for inferring that benzene has a six-membered ring.

3.1.2 OWL and SWRL Rules

The fundamental inability of OWL to represent cycles has motivated numerous research efforts towards enriching DL languages with the expressive power of rules [155, 105, 128, 129, 186, 77, 147]. In the current section we discuss SWRL (Semantic Web Rule Language) rules, which is one of the most well-known formalisms extending Description Logics with rules. Initially, we introduce the syntax and semantics of SWRL rules [105] and discuss their computational properties. Next, we present DL-safe rules [186], which is a fragment of SWRL rules that compromises some expressive power for decidability, and examine the applicability of DL-safe rules to the representation of structured objects.

Let $\mathcal{S}_{\mathsf{SWRL}} = (N_C, N_R, N_I, N_V)$ be a *SWRL signature* where N_C is a set of *concept names*, N_R is a set of *role names*, N_I is a set of *individual names* and N_V is a set of *variables*, such that N_C, N_R, N_I and N_V are countably infinite and pairwise disjoint

sets. A $\mathcal{SHOIQ}$ *knowledge base* is a $\mathcal{SROIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ such that role axioms are either transitivity axioms or RIAs of the form $R \sqsubseteq S$ for all roles R and S. Let $(\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{SHOIQ}$ knowledge base w.r.t. (N_C, N_R, N_I).The set of *SWRL atoms* w.r.t. $\mathcal{S}_{\mathsf{SWRL}}$ is the smallest set defined as follows:

1. $\mathsf{C(t)}$, where C is a concept and $\mathsf{t} \in N_I \cup N_V$.
2. $\mathsf{R(t, t')}$, where R is a role and $\mathsf{t}, \mathsf{t'} \in N_I \cup N_V$.
3. $\mathsf{t} \approx \mathsf{t'}$, where $\mathsf{t}, \mathsf{t'} \in N_I \cup N_V$.
4. $\mathsf{t} \not\approx \mathsf{t'}$, where $\mathsf{t}, \mathsf{t'} \in N_I \cup N_V$.

A *SWRL rule* is an expression of the form

$$\forall \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{m}.[\beta_1 \wedge \ldots \wedge \beta_\mathsf{n} \rightarrow \mathsf{h}_1 \wedge \ldots \wedge \mathsf{h}_\ell] \tag{3.6}$$

where $\mathsf{n}, \mathsf{m}, \ell \geq 0$, $\beta_1, \ldots, \beta_\mathsf{n}, \mathsf{h}_1, \ldots, \mathsf{h}_\ell$ are SWRL atoms, $\mathsf{Var}(\beta_1, \ldots, \beta_\mathsf{n}) = \{\mathsf{x_i}\}_{\mathsf{i}=1}^{\mathsf{m}}$ and $\mathsf{Var}(\mathsf{h}_1, \ldots, \mathsf{h}_\ell) = \mathsf{Var}(\beta_1, \ldots, \beta_\mathsf{n})$, where Var is defined over a set of SWRL atoms as expected. In addition to role axioms, concept inclusion axioms and assertions, a SWRL rule is also considered an axiom. A *SWRL knowledge base* is a quadruple of the form $\mathcal{K} = (\mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$, where $\mathcal{L}$ is a set of SWRL rules.

For an interpretation $\mathcal{I}$ and a SWRL rule r of the form (3.6) we say that $\mathcal{I}$ *satisfies* r (written $\mathcal{I} \models r$) if for each vector of domain elements $\vec{\mathsf{a}} = (\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{m})$ we have that $\mathcal{I} \models \{\beta_i\theta\}_{\mathsf{i}=1}^{\mathsf{m}}$ implies $\mathcal{I} \models \{\mathsf{h_i}\theta\}_{\mathsf{i}=1}^{\ell}$, where $\theta = \{\mathsf{x_i} \mapsto \mathsf{a_i}\}_{\mathsf{i}=1}^{\mathsf{m}}$, for $\mathsf{a}, \mathsf{b} \in N_I$ we have $\mathcal{I} \models \mathsf{a} \approx \mathsf{b}$ ($\mathcal{I} \models \mathsf{a} \not\approx \mathsf{b}$) if $\mathsf{a}^\mathcal{I} = \mathsf{b}^\mathcal{I}$ ($\mathsf{a}^\mathcal{I} \neq \mathsf{b}^\mathcal{I}$) and satisfaction of variable-free SWRL atoms by an interpretation $\mathcal{I}$ is the same as for ABox assertions. For a SWRL knowledge base $\mathcal{K} = (\mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$, we say that $\mathcal{I}$ is a model of $\mathcal{K}$ (written $\mathcal{I} \models \mathcal{K}$) if $\mathcal{I} \models (\mathcal{R}, \mathcal{T}, \mathcal{A})$ and $\mathcal{I} \models r$ for each $r \in \mathcal{L}$. For a SWRL knowledge base $\mathcal{K}$ and an axiom α, we say that $\mathcal{K}$ entails α (written $\mathcal{K} \models \alpha$) if for every model $\mathcal{I}$ of $\mathcal{K}$, we have $\mathcal{I} \models \alpha$.

The definition of the basic inference problems w.r.t. a SWRL knowledge base is analogous to the definition of inference problems for a $\mathcal{SROIQ}$ knowledge base. It has been shown that determining consistency of a SWRL knowledge base is undecidable

[128], as one can reduce the domino problem, which is known to be undecidable [25], to SWRL knowledge base consistency. In particular, the domino problem can be encoded as a SWRL knowledge base consistency problem by building an infinite two-dimensional grid using DL axioms and SWRL rules. Since there is provably no sound, complete and terminating algorithm for reasoning over a SWRL knowledge base, we do not consider SWRL rules to be a suitable formalism for the purposes of our scenarios.

In order to regain decidability, language fragments of SWRL rules have been identified where some expressivity is traded for decidability, such as DL-safe rules [186] and DL+safe rules [147]. We next focus on DL-safe rules which is a proper subset of SWRL; we restrict our presentation to rules with non-disjunctive heads, which are sufficient to assess the suitability of DL-safe rules for describing complex objects.

A $\mathcal{SHOIN}$ *knowledge base* is a $\mathcal{SHOIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ such that each 'at least', 'at most' and 'exactly' concept is of the form $(\geq n)S.\top$, $(\leq n)S.\top$ and $(= n)S.\top$ where $\top$ is the top concept. Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a $\mathcal{SHOIN}$ knowledge base w.r.t. (N_C, N_R, N_I). A *DL-safe rule* is an expression of the form

$$\forall \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{m}.[\mathsf{O}(\mathsf{x}_1) \wedge \ldots \wedge \mathsf{O}(\mathsf{x}_\mathsf{m}) \wedge \beta_1 \wedge \ldots \wedge \beta_\mathsf{n} \rightarrow \mathsf{h}_1 \wedge \ldots \wedge \mathsf{h}_\ell] \tag{3.7}$$

where the same restrictions as for a SWRL rule apply and additionally (i) each SWRL atom that occurs in (3.7) is either of the form $\mathsf{C}(\mathsf{t})$, where $\mathsf{C} \in N_C$, or of the form $\mathsf{R}(\mathsf{t}, \mathsf{t}')$, where R is a simple role (ii) $\mathsf{O} \in N_C$ and O does not occur in $\mathcal{K}$. As usual, we omit the universally quantified variables in the front to simplify the presentation. A $\mathcal{SHOIN}$ *DL-safe knowledge base* is a quadruple of the form $\mathcal{K} = (\mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$, where $\mathcal{L}$ is a set of DL-safe rules and $\{\mathsf{O}(\mathsf{i}) \mid \mathsf{i} \in \mathsf{Ind}(\mathcal{A})\} \subseteq \mathcal{A}$, where $\mathsf{Ind}(\mathcal{A})$ is the set of individuals that occur in $\mathcal{A}$. Please note that a DL-safe rule is a SWRL rule and a $\mathcal{SHOIN}$ DL-safe knowledge base is a SWRL knowledge base, so for a $\mathcal{SHOIN}$ DL-safe knowledge base $\mathcal{K}$ and an axiom α, the expression $\mathcal{K} \models \alpha$ is well-defined. Motik et al. [186] show that checking satisfiability of a $\mathcal{SHOIN}$ DL-safe knowledge base is decidable and Rosati [208] proves that the same problem is NExpTime-complete

(where DL-safe rules are of the form (3.7), i.e. without disjunction).

Intuitively the additional $\mathsf{O}(\mathsf{x})$ atoms in the body of each DL-safe rule ensure that inferences are restricted to individuals explicitly mentioned in the ABox of the knowledge base. On the one hand, this limitation secures decidability because it implies a bound on the number of atoms derivable by the DL-safe rules, but on the other hand it prevents the deduction of knowledge that refers to objects that are unknown or implied by existential quantifiers.

Now let us approach our example with the help of a $\mathcal{SHOIN}$ DL-safe knowledge base. Let r_1 and r_2 be two DL-safe rules, let α_4 and α_5 be two TBox axioms and let $\mathcal{R}_2$ be an RBox, defined as follows.

$$\begin{aligned}
&\bigwedge_{\mathsf{i}=0}^{6} \mathsf{O}(\mathsf{x_i}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x_0}, \mathsf{x_i}) \wedge \\
&\mathsf{Molecule}(\mathsf{x_0}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{bond}(\mathsf{x_i}, \mathsf{x}_{(\mathsf{i} \mod 6)+1}) \rightarrow \mathsf{MoleculeWithSixMemberedRing}(\mathsf{x_0}) \qquad (r_1)
\end{aligned}$$

$$\begin{aligned}
&\bigwedge_{\mathsf{i}=0}^{6} \mathsf{O}(\mathsf{x_i}) \wedge \mathsf{Molecule}(\mathsf{x_0}) \wedge \\
&\bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x_0}, \mathsf{x_i}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{Carbon}(\mathsf{x_i}) \wedge \\
&\bigwedge_{\mathsf{i}=1,3,5} \mathsf{singleBond}(\mathsf{x_i}, \mathsf{x}_{\mathsf{i}+1}) \wedge \\
&\bigwedge_{\mathsf{i}=2,4,6} \mathsf{doubleBond}(\mathsf{x_i}, \mathsf{x}_{(\mathsf{i} \mod 6)+1}) \rightarrow \mathsf{MoleculeWithBenzeneRing}(\mathsf{x_0}) \qquad (r_2)
\end{aligned}$$

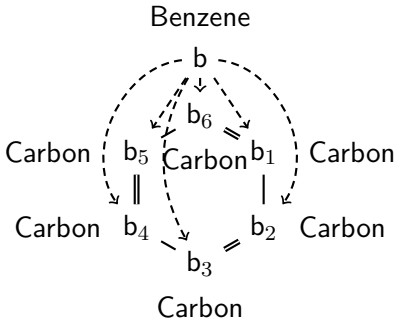
$$\mathsf{Benzene} \sqsubseteq \mathsf{Molecule} \qquad (\alpha_4)$$

$$\mathsf{Cyclobutane} \sqsubseteq \mathsf{Molecule} \qquad (\alpha_5)$$

$$\begin{aligned}
\mathcal{R}_2 = \{&\mathsf{singleBond} \sqsubseteq \mathsf{singleBond}^-, \mathsf{doubleBond} \sqsubseteq \mathsf{doubleBond}^-, \\
&\mathsf{singleBond} \sqsubseteq \mathsf{bond}, \mathsf{doubleBond} \sqsubseteq \mathsf{bond}\}
\end{aligned}$$

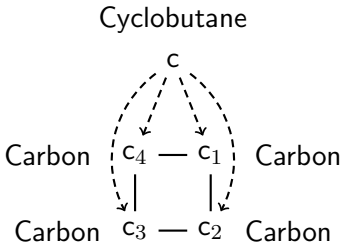
Consider the knowledge base $\mathcal{K}_4 = (\{r_1, r_2\}, \mathcal{R}_2, \{\alpha_4, \alpha_5\}, \mathcal{A}_\mathsf{b} \cup \mathcal{A}_\mathsf{c})$, where $\mathcal{A}_\mathsf{b}$ and $\mathcal{A}_\mathsf{c}$ are defined in Figure 3.3. By definition of SWRL semantics, we have

$$\mathcal{K}_4 \models \{\mathsf{MoleculeWithSixMemberedRing}(\mathsf{b}), \mathsf{MoleculeWithBenzeneRing}(\mathsf{b})\} \qquad (3.8)$$



(a) Graphical depiction of $\mathcal{A}_\mathsf{b}$

$$
\begin{aligned}
&\mathcal{A}_\mathsf{b} = \{\mathsf{Benzene(b)}\} \cup \\
&\{\mathsf{hasAtom(b, b_i), Carbon(b_i)}\}_{\mathsf{i}=1}^{6} \cup \\
&\{\mathsf{singleBond(b_i, b_{i+1}),}\}_{\mathsf{i}=1,3,5} \cup \\
&\{\mathsf{singleBond(b_{i+1}, b_i),}\}_{\mathsf{i}=1,3,5} \cup \\
&\{\mathsf{doubleBond(b_i, b_{(i \bmod 6)+1})}\}_{\mathsf{i}=2,4,6} \cup \\
&\{\mathsf{doubleBond(b_{(i \bmod 6)+1}, b_i)}\}_{\mathsf{i}=2,4,6}
\end{aligned}
$$



(b) Graphical depiction of $\mathcal{A}_\mathsf{c}$

$$
\begin{aligned}
&\mathcal{A}_\mathsf{c} = \{\mathsf{Cyclobutane(c)}\} \cup \\
&\{\mathsf{hasAtom(c, c_i), Carbon(c_i)}\}_{\mathsf{i}=1}^{4} \cup \\
&\{\mathsf{singleBond(b_i, b_{(i \bmod 4)+1})}\}_{\mathsf{i}=1}^{4} \cup \\
&\{\mathsf{singleBond(b_{(i \bmod 4)+1}, b_i)}\}_{\mathsf{i}=1}^{4}
\end{aligned}
$$

Figure 3.3: Definition of $\mathcal{A}_\mathsf{b}$ and $\mathcal{A}_\mathsf{c}$

Despite $\mathcal{K}_4$ entailing that individual b is a molecule with both a six-membered and a benzene ring, the described representation of benzene presents the following three drawbacks. First, we have

$$\mathcal{K}_4 \models \{\mathsf{MoleculeWithSixMemberedRing(c)}\} \tag{3.9}$$

which is obviously undesired, because c has only a four-membered ring. This is a result of the symmetry of bonds, which causes every individual with two atoms that are connected with a bond in both directions to have a six-membered ring. In order to prevent this effect, one needs to add inequality atoms—a feature not available for DL-safe rules—in the body of r_1 for the variables representing the atoms of the ring. Note that this is not problematic for r_2 because of the alternation of single and double bonds. Second, we infer the classes of a *particular instance* of benzene instead of the concept $\mathsf{Benzene}$, that is

$$\mathcal{K}_4 \not\models \mathsf{Benzene} \sqsubseteq \mathsf{MoleculeWithSixMemberedRing} \sqcap \mathsf{MoleculeWithBenzeneRing} \tag{3.10}$$

(3.10) means that we fail to classify the conceptual axiomatisation of benzene. To tackle this problem, one can check subsumption by adding $\mathsf{Benzene}(\mathsf{a})$, where a is a fresh individual, to the ABox and test whether $\mathcal{K}_4 \models \mathsf{MoleculeWithSixMemberedRing}(\mathsf{a})$ and $\mathcal{K}_4 \models \mathsf{MoleculeWithBenzeneRing}(\mathsf{a})$ hold. However, in that case we would have $\mathcal{K}_4 \not\models \mathsf{MoleculeWithSixMemberedRing}(\mathsf{a})$ and $\mathcal{K}_4 \not\models \mathsf{MoleculeWithBenzeneRing}(\mathsf{a})$, as there is no axiom that encodes the chemical structure of benzene. As a consequence, there are no guarantees that every object in the interpretation of benzene is also in the interpretation of molecules with a six-membered and a benzene ring, which clearly does not fulfill our requirements. Third, because of the first-order semantics of SWRL rules, we are still not able to model in a satisfactory manner hydrocarbon (see Section 3.1.1) and detect either its subclasses (e.g. $\mathsf{Benzene}, \mathsf{Cyclobutane}$) or its instances (e.g. b, c). In order to address the latter inadequacy of DL-safe rules, a formalism that extends Description Logic and rule knowledge bases with nonmonotonic negation has been suggested, known as *hybrid Minimal Knowledge and Negation as Failure (MKNF)* [185]. Nevertheless, hybrid MKNF draws upon DL-safe rules and, thus, inherits their limitation that constrains inferences only to the individuals asserted in the ABox.

3.1.3 OWL and Description Graphs

Description graphs (DG) are knowledge engineering constructs, which in combination with description logics and rules can provide a schema-level description of objects, whose components are arbitrarily interconnected [184]. Additionally, by applying suitable syntactic restrictions to the logic, reasoning tasks over DG-enriched knowledge bases can become decidable.

In this text, we refer to the description graphs framework presented by Motik et al. [184] as Description Graph Description Logics (DGDL). We adopt this name in order to distinguish DGDL from a DG-based syntax that we introduce in Chapter 7. In order to avoid misunderstandings, we also refer to the DGs described in this section as *FOL description graphs* (due to their first-order logic semantics) as opposed to simply *description graphs* that we use in Chapter 7.

Let $\mathcal{S}_{\mathsf{DGDL}} = (N_C, N_R, N_I, N_V)$ be a *DGDL signature* where N_C is a set of *concept names*, N_R is a set of *role names*, N_I is a set of *individual names* and N_V is a set of *variables*, such that N_C, N_R, N_I and N_V are countably infinite and pairwise disjoint. The set of *literal concepts* w.r.t. $\mathcal{S}_{\mathsf{DGDL}}$ is $N_L = N_C \cup \{\neg \mathsf{A} \mid \mathsf{A} \in N_C\}$. An *n-ary FOL description graph* G is a quadruple (V, E, λ, M) where $V = \{1, \ldots, n\}$ is a non-empty set of vertices, $E \subseteq V \times V$ is a set of edges, λ is a mapping that assigns to each vertex $i \in V$ a set of literal concepts $\lambda(i) \subseteq N_L$ and to each edge $(i, j) \in E$ a set of role names $\lambda(i, j) \subseteq N_R$ and, finally, $M \subseteq \bigcup_{i=1}^{n} \lambda(i)$ is a set of *main concepts* for G. A *graph box (GBox)* is a finite set of FOL description graphs. A *graph-enriched ABox* is an ABox such that assertions of the form $\neg\mathsf{R}(\mathsf{a}, \mathsf{b})$, where $\mathsf{R} \in N_R$ and $\mathsf{a}, \mathsf{b} \in N_I$, are disallowed and where assertions of the form $\mathsf{G}(\alpha_1, \ldots, \alpha_\mathsf{n})$, such that G is an n-ary description graph and $\{\alpha_i \mid 1 \leq i \leq \mathsf{n}\} \subseteq N_I$, can be added.

For an n-ary DG G, a *graph atom* is an expression of the form $\mathsf{G}(\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{n})$ where $\mathsf{t_i} \in N_I \cup N_V$ for $1 \leq \mathsf{i} \leq n$. A *graph rule* is an extension of a SWRL rule w.r.t. (N_C, N_R, N_I, N_V) such that graph atoms are allowed in the body and in the head of the rule, and all concepts and roles that occur in the rule are concept and role names, respectively. For a graph rule r, variables x and y are *directly connected* in r if they both occur in a body atom of r; *connected* is the transitive closure of directly connected. A graph rule is *connected* if each two variables that occur in r are connected.

A *DGDL knowledge base* is a quintuple $(\mathcal{G}, \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$, where $\mathcal{G}$ is a graph box, $\mathcal{L}$ is a finite set of connected graph rules, $(\mathcal{R}, \mathcal{T}, \emptyset)$ is a $\mathcal{SHOIQ}$ knowledge base and $\mathcal{A}$ is a graph-enriched ABox. A *graph-enhanced interpretation* $\mathcal{I}$ is an extension of an interpretation $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$, such that $\cdot^\mathcal{I}$ also maps each n-ary description graph $G \in N_G$ to an n-ary relation $G^\mathcal{I} \subseteq (\Delta^\mathcal{I})^n$. A graph-enhanced interpretation $\mathcal{I}$ satisfies a FOL description graph $G = (V, E, \lambda, M)$ (written $\mathcal{I} \models G$) if

1. For each $(a_1, \ldots, a_n) \in G^\mathcal{I}$ and $(b_1, \ldots, b_n) \in G^\mathcal{I}$ we have

$$\bigvee_{1 \leq i \leq n} a_i = b_i \rightarrow \bigwedge_{1 \leq j \leq n} a_j = b_j.$$

2. For each $(a_1, \ldots, a_n) \in G^\mathcal{I}$ and $(b_1, \ldots, b_n) \in G^\mathcal{I}$ we have $a_j \neq b_j$, where

$1 \leq i < j \leq n$.

3. For each concept name $A \in M$ and for each $x \in A^{\mathcal{I}}$ we have

$$\exists a_1, \ldots, a_n : (a_1, \ldots, a_n) \in G^{\mathcal{I}} \wedge \bigvee_{k \in V_A} x = a_k$$

where $V_A = \{i \in V \mid A \in \lambda(i)\}$.

4. For each $(a_1, \ldots, a_n) \in G^{\mathcal{I}}$ we have

$$\bigwedge_{i \in V, B \in \lambda(i)} a_i \in B^{\mathcal{I}} \wedge \bigwedge_{(i,j) \in E, R \in \lambda(i,j)} (a_i, a_j) \in R^{\mathcal{I}}$$

.

A graph-enhanced interpretation $\mathcal{I}$ satisfies a graph assertion $G(\alpha_1, \ldots, \alpha_n)$ if $(\alpha_1^{\mathcal{I}}, \ldots, \alpha_n^{\mathcal{I}}) \in G^{\mathcal{I}}$. Satisfaction of a connected graph rule by a graph-enhanced interpretation $\mathcal{I}$ is defined analogously to satisfaction of a SWRL rule by an interpretation $\mathcal{I}$. Additionally, a graph-enhanced interpretation $\mathcal{I}$ satisfies a DGDL knowledge base $(\mathcal{G}, \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$ if $\mathcal{I}$ satisfies each FOL description graph in $\mathcal{G}$, each connected graph rule in $\mathcal{L}$, each graph assertion in $\mathcal{A}$ and, also, $\mathcal{I}$ satisfies the $\mathcal{SHOIQ}$ knowledge base $(\mathcal{R}, \mathcal{T}, \mathcal{A}_{\mathsf{DL}})$, where $\mathcal{A}_{\mathsf{DL}}$ is the maximal subset of $\mathcal{A}$ with no graph assertions. Finally, entailment of (a set) of axioms by a DGDL knowledge base and consistency of a DGDL knowledge base are defined as usual.

Determining satisfiability of a DGDL knowledge base $\mathcal{K} = (\mathcal{G}, \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$ is undecidable; the problem remains undecidable even when one considers $\mathcal{K}$ with $\mathcal{L} = \emptyset$ or when one considers $\mathcal{K}$ with $\mathcal{T} \cup \mathcal{R} = \emptyset$ [184]. In order to regain decidability of reasoning, Motik et al. suggest restrictions on the syntax of a DGDL knowledge base [184]. In particular, an *acyclicity* and a *strong separation* condition are proposed which we present next.

Let $\mathcal{K} = (\mathcal{G}, \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{A})$ be a DGDL knowledge base. Let $G_1 \in \mathcal{G}$ and $G_2 \in \mathcal{G}$ be an n-ary and an m-ary FOL description graph, respectively, such that $\mathsf{n} \leq \mathsf{m}$; let $G_1 \lhd G_2$ if there exists a rule in $\mathcal{L}$ of the form

$$\mathsf{G}_2(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}, \ldots, \mathsf{x}_\mathsf{m}) \rightarrow \mathsf{G}_1(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n})$$

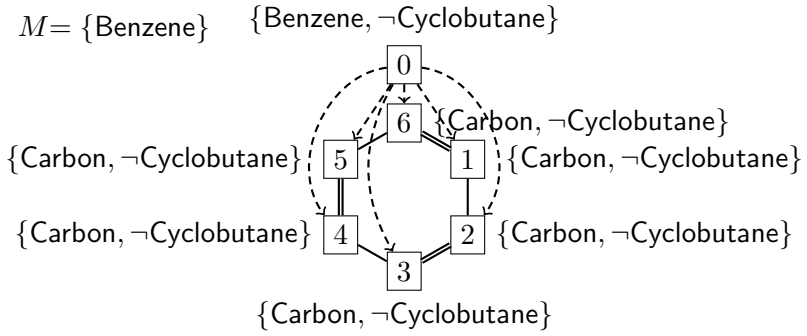
and let $\trianglelefteq$ be the reflexive and transitive closure of $\triangleleft$. The knowledge base $\mathcal{K}$ is *acyclic* if an irreflexive and transitive order $\prec$ on $\mathcal{G}$ exists such that, for each $G = (V, E, \lambda, M)$ and $G' = (V', E', \lambda', M')$ where $G, G' \in \mathcal{G}$ and $G \not\trianglelefteq G'$ and for each $A' \in M'$, we have

- $G' \trianglelefteq G$ implies $\neg A' \in \lambda(i)$ for each $i \in V \setminus V'$ and
- $G' \not\trianglelefteq G$ implies $\neg A' \in \lambda(i)$ for each $i \in V$.

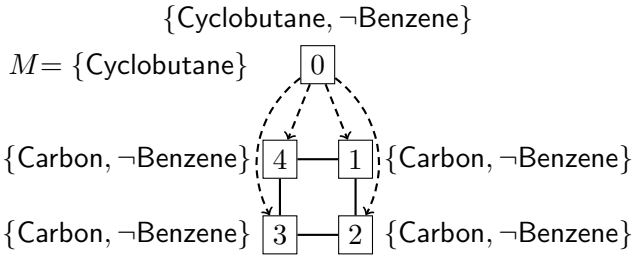
Furthermore, let $N_R^{\mathcal{G}}$, $N_R^{\mathcal{L}}$ and $N_R^{\mathcal{R},\mathcal{T}}$ be the set of role names that occur in $\mathcal{G}$, in $\mathcal{L}$ and in $\mathcal{R}$ and $\mathcal{T}$, respectively. The knowledge base $\mathcal{K}$ is *strongly separated* if $(N_R^{\mathcal{G}} \cup N_R^{\mathcal{L}}) \cap N_R^{\mathcal{R},\mathcal{T}} = \emptyset$.

The satisfiability problem of an acyclic and strongly separated DGDL knowledge base is decidable [184]. Intuitively, the acyclicity condition ensures that the GBox has an interpretation of finite size by requiring entailment of a contradiction for each model that contains graph instances repeated in a cyclic way. On the other hand, the strong separation condition prevents any interaction between the GBox definitions and the TBox axioms that may cause the existence of unbounded arbitrarily shaped models of the knowledge base, which in turn can result in undecidability of the main reasoning tasks.

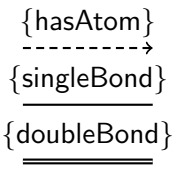
Since DGDL knowledge bases allow for modelling non-tree-like structures on the schema-level, i.e. not restricted to the individuals explicitly named in the ABox, one can apply them to the representation of structured entities. Returning to the example of structure-based chemical classification, the chemical graphs of benzene and cyclobutane can be described with the FOL description graphs $\mathsf{G}_{\mathsf{benzene}}$ and $\mathsf{G}_{\mathsf{cyclobutane}}$ that appear in Figure 3.4, where the labellings of the nodes appear in curly brackets and the labellings of the edges are provided by the key. The negative literals in the labellings of the nodes are required to make the knowledge base acyclic by ensuring the existence of a suitable order $\prec$ (which in this case is the empty relation). Additionally, the following connected graph rules can be used to model the class of molecules that contain certain types of rings as well as the subsumption relations for single and double bonds. Please note that rules r_5 and r_6 cannot be expressed as role inclusion axioms of the RBox because of the strong separation



(a) FOL description graph $\mathsf{G}_{\mathsf{benzene}}$



(b) FOL description graph $\mathsf{G}_{\mathsf{cyclobutane}}$



(c) Edge labels key

Figure 3.4: Definition of $\mathsf{G}_{\mathsf{benzene}}$ and $\mathsf{G}_{\mathsf{cyclobutane}}$

condition.

$$\mathsf{Molecule}(\mathsf{x}_0) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x}_0, \mathsf{x}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{bond}(\mathsf{x}_\mathsf{i}, \mathsf{x}_{(\mathsf{i} \mod 6)+1}) \rightarrow \mathsf{MoleculeWithSixMemberedRing}(\mathsf{x}_0) \qquad (r_3)$$

$$\mathsf{Molecule}(\mathsf{x}_0) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x}_0, \mathsf{x}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{Carbon}(\mathsf{x}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1,3,5} \mathsf{singleBond}(\mathsf{x}_\mathsf{i}, \mathsf{x}_{\mathsf{i}+1}) \wedge \bigwedge_{\mathsf{i}=2,4,6} \mathsf{doubleBond}(\mathsf{x}_\mathsf{i}, \mathsf{x}_{(\mathsf{i} \mod 6)+1}) \rightarrow \mathsf{MoleculeWithBenzeneRing}(\mathsf{x}_0) \qquad (r_4)$$

$$\mathsf{singleBond}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{bond}(\mathsf{x}, \mathsf{y}) \qquad (r_5)$$

$$\mathsf{doubleBond}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{bond}(\mathsf{x}, \mathsf{y}) \qquad (r_6)$$

From the DGDL knowledge base $\mathcal{K}_5 = (\{\mathsf{G}_{\mathsf{benzene}}, \mathsf{G}_{\mathsf{cyclobutane}}\}, \{r_i\}_{i=3}^{6}, \emptyset, \{\alpha_4, \alpha_5\}, \emptyset)$,

it can be checked that the following subclass relations follow.

$$\mathcal{K}_5 \models \mathsf{Benzene} \sqsubseteq \mathsf{MoleculeWithSixMemberedRing}$$

$$\mathcal{K}_5 \models \mathsf{Benzene} \sqsubseteq \mathsf{MoleculeWithBenzeneRing}$$

It is important to observe that the above subsumption axiom for benzene is entailed by a knowledge base with an empty ABox, that is we have succeeded to classify the *concept* representing the benzene molecule rather than a specific instance of it.

The DGDL approach advances the state of the art for the representation of structured objects; however, it also presents some weaknesses that we discuss next. First, since inequality atoms are not permitted in the body of connected graph rules, we are not able to specify in rule r_3 that the atoms of a six-membered ring need to be distinct, which gives rise to the following undesired entailment:

$$\mathcal{K}_5 \models \mathsf{Cyclobutane} \sqsubseteq \mathsf{MoleculeWithSixMemberedRing}$$

A second disadvantage is the strong separation restriction that prevents the modeller from mixing roles used in the TBox with roles occurring in the GBox or the set of rules. This constraint can limit the applicability of the formalism as, for example, axiom α_6 cannot be added to the TBox of $\mathcal{K}_5$.

$$\exists \mathsf{hasAtom}.(\mathsf{Oxygen} \sqcap \exists \mathsf{bond}.\mathsf{Hydrogen}) \sqsubseteq \exists \mathsf{hasFunctionalGroup}.\mathsf{HydroxyGroup} \quad (\alpha_6)$$

Axiom α_6 is essential in order to infer, e.g. chemical classes of molecules depending on containment of functional groups [234]; functional groups characterise molecules in terms of chemical reactivity and are thus important molecular descriptors. For instance, with the additional axiom

$$\exists \mathsf{hasFunctionalGroup}.\mathsf{HydroxyGroup} \sqsubseteq \mathsf{Alcohol} \quad (\alpha_7)$$

one could build a knowledge base $\mathcal{K}_6 = (\{\mathsf{G}_{\mathsf{benzene}}, \mathsf{G}_{\mathsf{cyclobutane}}\}, \{r_i\}_{i=3}^{6}, \emptyset, \{\alpha_i\}_{i=6}^{7}, \emptyset)$ that would entail the following subsumption.

$$\exists \mathsf{hasAtom}.(\mathsf{Oxygen} \sqcap \exists \mathsf{bond}.\mathsf{Hydrogen}) \sqsubseteq \mathsf{Alcohol} \quad (\alpha_8)$$

However, this is not feasible, because $\mathcal{K}_6$ violates the strong separation requirement. Third, as a consequence of the first-order logic semantics of the DGDL formalism and similarly to the case of a $\mathcal{SROIQ}$ and a $\mathcal{SHOIQ}$ DL-safe knowledge base, one is not able to model classes that are based on the absence of certain characteristics; for example, one cannot effectively recognise chemical classes such as hydrocarbons or inorganic molecules. Fourth, the mentioned acyclicity condition lacks practicality because it requires the modeller to extend the labellings of the DG nodes with a—potentially large—number of negative literals. Computing the appropriate negative literals can complicate the construction of FOL description graphs—although this problem could be resolved by suitable ontology development tool support. Finally, by the definition of the acyclicity condition, models containing a cyclic chain of description graph instances are precluded by making the knowledge base inconsistent, which makes it difficult to distinguish whether the inconsistency of a knowledge base stems from a conceptual modelling error or from the acyclicity check.

3.1.4 Further Rule-Based Approaches

An additional language that integrates DL axioms with rules containing nonmonotonic negation is dl-programs [77]. Nonetheless, no function symbols are allowed in the structure of the rules which implies the same restrictions as the ones imposed by DLs and their extensions with DL-safe rules. In the same spirit, Eiter et al. consider a nonmonotonic rule-based ontology language under stable model semantics, but with no use of function symbols either [78].

A knowledge representation formalism that combines nonmonotonic negation and rules but allows use of function symbols (function symbols can be used to simulate existential restrictions of Description Logics) is $\mathbb{FDNC}$ programs [79]. In order to ensure decidability of reasoning over $\mathbb{FDNC}$ programs, the structure of rules is restricted to one of seven predefined forms. However, rules that are essential for the representation of structures (such as the rules describing the cyclic atom arrangement of cyclobutane) violate these restrictions which makes $\mathbb{FDNC}$ programs not suitable for the purposes of our modelling.

Nonmonotonic extensions of rules with existentially quantified variabels in the head have also been considered using stratified negation [40, 174] and more recently using well-founded semantics [96]. In the latter, the so-called *equality-friendly well-founded semantics* for nonmonotonic existential rules is defined and complexity results are provided for query answering over guarded rules interpreted under the newly defined semantics. However, as we explain in Section 3.3.1 non-guarded rules are essential for modelling in our setting.

Although we have discussed the abilities of several logic-based frameworks to represent complex structures, no exhaustive literature survey on all rule-based ontology languages is provided here; such overviews can be found in the works by Krötzsch [147] and Motik and Rosati [185].

3.2 Nonmonotonic Existential Rules as a Description Language

After having examined a number of well-established knowledge representation formalisms for modelling structured entities, we suggest a completely different approach that draws upon *nonmonotonic existential rules*. We demonstrate how the structures discussed in the examples of the previous section can be described and automatically classified with the help of logic programs consisting of nonmonotonic existential rules and how the obstacles resulting from the expressivity limitations of OWL and its extensions can be overcome. In a nutshell, the methodology we suggest is to represent knowledge about structured objects with a program comprising rules of the form (2.1) and, then, identify the entailed subsumptions by checking the stable models of a suitably extended program, as defined in Section 2.3.

One can use rules r_7' and r_8' next to represent the chemical graphs of benzene and

cyclobutane respectively.

$$\mathsf{benzene}(\mathsf{x}) \rightarrow \exists_{\mathsf{i}=1}^{6}\mathsf{y}_\mathsf{i}.\mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x},\mathsf{y}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{carbon}(\mathsf{x}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1,3,5} \mathsf{singleBond}(\mathsf{x}_\mathsf{i},\mathsf{x}_{\mathsf{i}+1}) \wedge \bigwedge_{\mathsf{i}=2,4,6} \mathsf{doubleBond}(\mathsf{x}_\mathsf{i},\mathsf{x}_{(\mathsf{i} \mod 6)+1}) \quad (r_7')$$

$$\mathsf{cyclobutane}(\mathsf{x}) \rightarrow \exists_{\mathsf{i}=1}^{6}\mathsf{y}_\mathsf{i}.\mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{hasAtom}(\mathsf{x},\mathsf{y}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{carbon}(\mathsf{x}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{singleBond}(\mathsf{x}_\mathsf{i},\mathsf{x}_{\mathsf{i}+1}) \quad (r_8')$$

The skolemised versions of r_7' and r_8' are as follows.

$$\mathsf{benzene}(\mathsf{x}) \rightarrow \mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x},\mathsf{f}_\mathsf{i}(\mathsf{x})) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{carbon}(\mathsf{f}_\mathsf{i}(\mathsf{x})) \wedge \bigwedge_{\mathsf{i}=1,3,5} \mathsf{singleBond}(\mathsf{f}_\mathsf{i}(\mathsf{x}),\mathsf{f}_{\mathsf{i}+1}(\mathsf{x})) \wedge \bigwedge_{\mathsf{i}=2,4,6} \mathsf{doubleBond}(\mathsf{f}_\mathsf{i}(\mathsf{x}),\mathsf{f}_{(\mathsf{i} \mod 6)+1}(\mathsf{x})) \quad (r_7)$$

$$\mathsf{cyclobutane}(\mathsf{x}) \rightarrow \mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{hasAtom}(\mathsf{x},\mathsf{g}_\mathsf{i}(\mathsf{x})) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{carbon}(\mathsf{g}_\mathsf{i}(\mathsf{x})) \wedge \bigwedge_{\mathsf{i}=1}^{4} \mathsf{singleBond}(\mathsf{g}_\mathsf{i}(\mathsf{x}),\mathsf{g}_{\mathsf{i}+1}(\mathsf{x})) \quad (r_8)$$

Furthermore, we can exploit the nonmonotonic negation of the stable model semantics to model hydrocarbons, that is the class of molecules that consist only of hydrogens or carbons and contain at least one carbon atom.

$$\mathsf{hasAtom}(\mathsf{x},\mathsf{y}) \wedge \mathsf{carbon}(\mathsf{y}) \rightarrow \mathsf{carbonMolecule}(\mathsf{x}) \quad (r_9)$$

$$\mathsf{hasAtom}(\mathsf{x},\mathsf{y}) \wedge \ \mathbf{not}\ \mathsf{carbon}(\mathsf{y}) \wedge \ \mathbf{not}\ \mathsf{hydrogen}(\mathsf{y}) \rightarrow \mathsf{notHydrocarbon}(\mathsf{x}) \quad (r_{10})$$

$$\mathsf{carbonMolecule}(\mathsf{x}) \wedge \ \mathbf{not}\ \mathsf{notHydroCarbon}(\mathsf{x}) \rightarrow \mathsf{hydroCarbon}(\mathsf{x}) \quad (r_{11})$$

One can model molecules that contain a six-membered ring with a rule similar to r_3. Thanks to the fact that rules allow for negated atoms in the body, we are able to add inequality atoms in the body of r_3 and, thus, prevent molecules that contain a

pair of atoms connected with a bond from being classified as molecules with a six-membered ring. We define equality as a predicate $=$, such that (i) for every constant c in the program or set of facts, the set of facts also contains $\mathsf{c} = \mathsf{c}$ and (ii) for every existentially quantified variable y in the head of a rule, the head also contains the atom $\mathsf{y} = \mathsf{y}$. Since all equality atoms are identities, we do not need to axiomatise properties of equality such as replacement, transitivity or symmetry. In the rest of this thesis, we treat equality as implicitly defined and do not include it in sets of facts, stable models and rule heads; we also abbreviate inequality atoms of the form **not** $\mathsf{x} = \mathsf{y}$ with $\mathsf{x} \neq \mathsf{y}$. We are now ready to define rule r_{12} for molecules with a six-membered ring.

$$\mathsf{Molecule}(\mathsf{x}_0) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x}_0, \mathsf{x}_\mathsf{i}) \wedge \bigwedge_{1 \leq \mathsf{i} < \mathsf{j} \leq 6} \mathsf{x}_\mathsf{i} \neq \mathsf{x}_\mathsf{j} \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{bond}(\mathsf{x}_\mathsf{i}, \mathsf{x}_{(\mathsf{i} \mod 6)+1}) \rightarrow \mathsf{MoleculeWithSixMemberedRing}(\mathsf{x}_0) \quad (r_{12})$$

Finally, we can reuse rules r_4– r_6 (page 41) to model molecules with a benzene ring as well as the subproperty relation for single and double bonds. Inequality atoms are not necessary in the body of rule r_4 because of the alternation of single and double bonds. Finally, we add rules r_{13} and r_{14} to establish symmetry of single and double bonds.

$$\mathsf{singleBond}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{singleBond}(\mathsf{y}, \mathsf{x}) \quad (r_{13})$$

$$\mathsf{doubleBond}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{doubleBond}(\mathsf{y}, \mathsf{x}) \quad (r_{14})$$

Let $P = \{r_i \mid 4 \leq i \leq 14\}$. We can derive the subsumers of $\mathsf{benzene}$ by extending P with the fact $\mathsf{benzene}(\mathsf{b})$ and identify in the intersection of stable models of $P \cup \{\mathsf{benzene}(\mathsf{b})\}$ all the unary atoms that have b as their argument (similarly for $\mathsf{cyclobutane}$). It is easy to verify that $P \cup \{\mathsf{benzene}(\mathsf{b})\}$ has only one stable model,

which is $\mathcal{M}_{\mathsf{b}}$.

$$\begin{aligned}\mathcal{M}_{\mathsf{b}} =&\{\mathsf{benzene}(\mathsf{b})\} \cup \{\mathsf{hasAtom}(\mathsf{b}, \mathsf{f_i}(\mathsf{b})), \mathsf{carbon}(\mathsf{f_i}(\mathsf{b}))\}_{\mathsf{i}=1}^{6} \cup \\ &\{\mathsf{singleBond}(\mathsf{f_i}(\mathsf{b}), \mathsf{f_{i+1}}(\mathsf{b})), \mathsf{singleBond}(\mathsf{f_{i+1}}(\mathsf{b}), \mathsf{f_i}(\mathsf{b}))\}_{\mathsf{i}=1,3,5} \cup \\ &\{\mathsf{doubleBond}(\mathsf{f_i}(\mathsf{b}), \mathsf{f}_{(\mathsf{i} \mod 6)+1}(\mathsf{b})), \mathsf{doubleBond}(\mathsf{f_{i+1}}(\mathsf{b}), \mathsf{f_i}(\mathsf{b}))\}_{\mathsf{i}=2,4,6} \cup \\ &\{\mathsf{bond}(\mathsf{f_i}(\mathsf{b}), \mathsf{f}_{(\mathsf{i} \mod 6)+1}(\mathsf{b})), \mathsf{bond}(\mathsf{f}_{(\mathsf{i} \mod 6)+1}(\mathsf{b}), \mathsf{f_i}(\mathsf{b}))\}_{\mathsf{i}=1}^{6} \cup \\ &\{\mathsf{molecule}(\mathsf{b}), \mathsf{carbonMolecule}(\mathsf{b}), \mathsf{hydrocarbon}(\mathsf{b}), \} \cup \\ &\{\mathsf{moleculeWithBenzeneRing}(\mathsf{b}), \mathsf{moleculeWithSixMemberedRing}(\mathsf{b})\}\end{aligned}$$

From the atoms contained in $\mathcal{M}_{\mathsf{b}}$, entailments (3.11)–(3.15) follow.

$$P \models \mathsf{benzene} \sqsubseteq \mathsf{molecule} \tag{3.11}$$

$$P \models \mathsf{benzene} \sqsubseteq \mathsf{carbonMolecule} \tag{3.12}$$

$$P \models \mathsf{benzene} \sqsubseteq \mathsf{hydrocarbon} \tag{3.13}$$

$$P \models \mathsf{benzene} \sqsubseteq \mathsf{moleculeWithSixMemberedRing} \tag{3.14}$$

$$P \models \mathsf{benzene} \sqsubseteq \mathsf{moleculeWithBenzeneRing} \tag{3.15}$$

Similarly for cyclobutane, the stable model and the entailed subsumptions appear next.

$$\begin{aligned}\mathcal{M}_{\mathsf{c}} =&\{\mathsf{cyclobutane}(\mathsf{c})\} \cup \{\mathsf{hasAtom}(\mathsf{c}, \mathsf{g_i}(\mathsf{c})), \mathsf{carbon}(\mathsf{g_i}(\mathsf{c}))\} \cup \\ &\{\mathsf{singleBond}(\mathsf{g}_{(\mathsf{i} \mod 4)+1}(\mathsf{c}), \mathsf{g_i}(\mathsf{c})), \mathsf{singleBond}(\mathsf{g_i}(\mathsf{c}), \mathsf{g}_{(\mathsf{i} \mod 4)+1}(\mathsf{c}))\}_{\mathsf{i}=1}^{4} \cup \\ &\{\mathsf{bond}(\mathsf{g}_{(\mathsf{i} \mod 4)+1}(\mathsf{c}), \mathsf{g_i}(\mathsf{c})), \mathsf{bond}(\mathsf{g_i}(\mathsf{c}), \mathsf{g}_{(\mathsf{i} \mod 4)+1}(\mathsf{c}))\}_{\mathsf{i}=1}^{4} \cup \\ &\{\mathsf{molecule}(\mathsf{c}), \mathsf{carbonMolecule}(\mathsf{c}), \mathsf{hydrocarbon}(\mathsf{c})\}\end{aligned}$$

$$P \models \mathsf{cyclobutane} \sqsubseteq \mathsf{molecule}$$

$$P \models \mathsf{cyclobutane} \sqsubseteq \mathsf{carbonMolecule}$$

$$P \models \mathsf{cyclobutane} \sqsubseteq \mathsf{hydrocarbon}$$

By adopting nonmonotonic existential rules, we tackle many of the problems outlined in the previous section. First, the use of negation-as-failure enables the definition of

classes that are difficult or complicated to define using formalisms equipped with first-order logic semantics, such as OWL or its extensions with DL-safe rules and description graphs. Second, we are capable of representing and recognising cyclic structures with precision without, for instance, overestimating the subsumees of structures with a ring of fixed length. Additionally, there is no need for a separation condition that enforces use of binary predicates only in certain types of rules, e.g. we can model the knowledge captured by axioms α_6 and α_7 using rules. Finally, in the framework presented so far there is no need for an acyclicity condition that requires addition of extra negative literals labelling the nodes of the objects that we represent.

In the following section, we detail complications that emerge from the use of nonmonotonic existential rules as an ontology language and approaches that have been taken in the past to remedy them.

3.3 Difficulties of Nonmonotonic Existential Rules

A number of challenges that show up when using nonmonotonic existential rules as a knowledge representation formalism appears next. First, every knowledge representation language requires a formally defined procedure for automated reasoning. For the case of nonmonotonic existential rules, a sound, complete and terminating algorithm for stable model computation is needed in order to check the entailed subsumptions. There are two main difficulties when developing such algorithms: one is related to the calculation and representation of stable models of infinite size and the other to the existence of more than one stable model; we discuss both problems next. An additional problem that we identify is that quite often natural definitions require using multiple non-intuitive rules, e.g. rules r_{10} and r_{11} for the class of hydrocarbons, which hinders adoption by domain experts, who are the professionals that mainly develop ontologies. In the remainder of this section, we examine these difficulties in more detail and review approaches that have been taken to handle them.

3.3.1 Stable Models of Infinite Size

We next turn our attention to stable model computation for sets of nonmonotonic existential rules and how this computation becomes impractical for stable models of infinite size. Initially, we study the problem in the context of positive logic programs and we also discuss the approaches taken for rules with non-empty negative bodies.

Chase [4] is the standard forward-chaining algorithmic procedure for performing bottom-up computation of stable models for existential rules. The chase algorithm was introduced in the late 1970s for query answering under database dependencies [134, 176, 24] and bears a strong resemblance to the tableau reasoning algorithm [85], one of the standard algorithms for DL reasoning [125, 12]. The main idea behind chase which operates under a set of rules is to extend the initial set of facts with suitably instantiated atoms so that no rule is further applicable. Several variants of the chase have been suggested, including the *restricted chase* (also known and as nonoblivious chase [176, 80]), where a rule is applicable only if its head is not satisfied by the set of facts derived so far, the *oblivious chase* [134, 39], where such a check is not required, and the *skolem chase* [177, 99], where chase is defined as a fixed point computation over the skolemised version of the rules. In the former two cases when adding instantiated rule heads existentially quantified variables are replaced by fresh constants (an infinite supply of which is assumed), whereas in the latter case function terms are added instead due to skolemisation. In the current thesis, we adopt the skolem chase because of its intuitive declarative definition and of the advantage that it can capture in the structure of functional terms the history of rule application. Next, we formally introduce the chase procedure.

Definition 4 (Skolem Chase). *For a skolemised positive rule $r = (B^+, \emptyset, H)$ and a set of facts F, the* application *of r to F, written $r(F)$, is defined as*

$$r(F) = \bigcup_{B^+\theta \subseteq F} H\theta$$

where θ is a substitution from $\mathsf{Var}(r)$ to $\mathsf{Const}(F)$. For a set of skolemised positive rules P, we define $P(F) = \bigcup_{r \in P} r(F)$. Let F be a set of facts, let P be a set of positive rules and let P_d be the set of datalog rules in P and let $P_f = P \setminus P_d$. The

chase sequence *of* P *w.r.t.* F *is a sequence of sets of facts* F_P^0, F_P^1, ... *such that* $F_P^0 = F$ *and for* $i \geq 0$

$$F_P^{i+1} = \begin{cases} F_P^i \cup P_d(F_P^i), & \text{if } P_d(F_P^i) \not\subseteq F_P^i \\ F_P^i \cup P_f(F_P^i), & \text{otherwise.} \end{cases}$$

The chase *of* P *w.r.t.* F *is* $\mathsf{Chase}_P(F) = \bigcup_{i \geq 0} F_P^i$.

It is straightforward to see that the chase of $\mathsf{sk}(P)$ w.r.t. F coincides with the (unique) stable model of $\mathsf{sk}(P) \cup F$.

Example 5. Assume that one wants to model chemical knowledge about toxicity of molecules, such as botox,[1] using the following rules.

$$\mathsf{botox}(\mathsf{x}_1) \rightarrow \mathsf{highlyToxic}(\mathsf{x}_1) \wedge \mathsf{metabolite}(\mathsf{x}_1) \quad (r_1)$$

$$\mathsf{highlyToxic}(\mathsf{x}_2) \rightarrow \exists \mathsf{y}_2.\mathsf{hasProperty}(\mathsf{x}_2, \mathsf{y}_2) \wedge \mathsf{highToxicity}(\mathsf{y}_2) \quad (r_2)$$

$$\mathsf{hasProperty}(\mathsf{x}_3, \mathsf{z}_3) \wedge \mathsf{highToxicity}(\mathsf{z}_3) \rightarrow \mathsf{highlyToxic}(\mathsf{x}_3) \quad (r_3)$$

$$\mathsf{highToxicity}(\mathsf{x}_4) \rightarrow \exists \mathsf{y}_4.\mathsf{hasToxicityDegree}(\mathsf{x}_4, \mathsf{y}_4) \wedge \mathsf{toxicityDegreeOne}(\mathsf{y}_4) \quad (r_4)$$

$$\mathsf{hasToxicityDegree}(\mathsf{x}_5, \mathsf{w}_5) \rightarrow \mathsf{hasProperty}(\mathsf{x}_5, \mathsf{w}_5) \quad (r_5)$$

$$\mathsf{metabolite}(\mathsf{x}_6) \wedge \mathsf{hasProperty}(\mathsf{x}_6, \mathsf{z}_6) \wedge \mathsf{hasToxicityDegree}(\mathsf{z}_6, \mathsf{w}_6) \rightarrow \mathsf{toxin}(\mathsf{x}_6) \quad (r_6)$$

$$\mathsf{toxin}(\mathsf{x}_7) \wedge \mathsf{hasProperty}(\mathsf{x}_7, \mathsf{z}_7) \wedge \mathsf{hasProperty}(\mathsf{z}_7, \mathsf{w}_7) \wedge \mathsf{toxicityDegreeOne}(\mathsf{w}_7) \rightarrow \mathsf{lethalToxin}(\mathsf{x}_7) \quad (r_7)$$

Let f and g be the function symbols used to skolemise y_2 and y_4 respectively, let $P_\mathsf{b} = \{r_1, \mathsf{sk}(r_2), r_3, \mathsf{sk}(r_4), r_5, r_6, r_7\}$ and let $F = \{\mathsf{botox}(\mathsf{b})\}$. It follows that

[1] Botox is the trade name of botulinum toxin, the most acutely toxic substance known [11].

$$F^0_{P_\mathsf{b}} = F$$

$$F^1_{P_\mathsf{b}} = F^0_{P_\mathsf{b}} \cup \{\mathsf{highlyToxic(b)}, \mathsf{metabolite(b)}\}$$

$$F^2_{P_\mathsf{b}} = F^1_{P_\mathsf{b}} \cup \{\mathsf{hasProperty(b, f(b))}, \mathsf{highToxicity(f(b))}\}$$

$$F^3_{P_\mathsf{b}} = F^2_{P_\mathsf{b}} \cup \{\mathsf{hasToxicityDegree(f(b), g(f(b)))}, \mathsf{toxicityDegreeOne(g(f(b)))}\}$$

$$F^4_{P_\mathsf{b}} = F^3_{P_\mathsf{b}} \cup \{\mathsf{hasProperty(f(b), g(f(b)))}, \mathsf{toxin(b)}\}$$

$$F^5_{P_\mathsf{b}} = F^4_{P_\mathsf{b}} \cup \{\mathsf{lethalToxin(b)}\} = \mathsf{Chase}_{P_\mathsf{b}}(F)$$

$\mathsf{Chase}_{P_\mathsf{b}}(F)$ is the only stable model of $P_\mathsf{b} \cup F$ from which we can infer the following entailments:

$$P_\mathsf{b} \models \mathsf{botox} \sqsubseteq \mathsf{highlyToxic}$$

$$P_\mathsf{b} \models \mathsf{botox} \sqsubseteq \mathsf{metabolite}$$

$$P_\mathsf{b} \models \mathsf{botox} \sqsubseteq \mathsf{toxin}$$

$$P_\mathsf{b} \models \mathsf{botox} \sqsubseteq \mathsf{lethalToxin}$$

♢

One can easily see that for a logic program P, if P consists only of datalog rules, then there is a bound on the size of $\mathsf{Chase}_{P_\mathsf{b}}(F)$. However, if P contains even one existential rule, $\mathsf{Chase}_{P_\mathsf{b}}(F)$ can be of infinite size as Example 6 shows.

Example 6. Let $F = \{\mathsf{person(alice)}\}$ and let r be defined as follows.

$$\mathsf{person(x)} \rightarrow \exists \mathsf{y}_1, \mathsf{y}_2.\mathsf{hasMother(x, y_1)} \wedge \mathsf{person(y_1)} \wedge \mathsf{hasFather(x, y_2)} \wedge \mathsf{person(y_2)} \qquad (r)$$

$\mathsf{Chase}_{\{\mathsf{sk}(r)\}}(F)$ contains infinitely many facts. ♢

Therefore, a decision problem that naturally arises is, given a set of positive rules P and a set of facts F, is $\mathsf{Chase}_P(F)$ finite? It has been proved that this decision problem, also known and as *chase termination*, is undecidable [71]. Moreover, the problem of query answering under arbitrary existential rules is undecidable [24, 53].

As a consequence, an extensive line of research has been dedicated to identifying decidable fragments of existential rules. Due to the applicability of existential rules to a wide range of modelling problems, relevant contributions have emerged from a variety of domains such as theory of databases, logic programming and rule-based knowledge representation. Techniques for ensuring decidability are usually inspired by one of three different paradigms, each of which is associated with a generic property that characterises sets of rules and that is not decidable to recognise [42, 17]. A discussion of these three paradigms follows; the first two paradigms that we discuss suggest conditions that confine the structure of the rules so that their—potentially of infinite size—models have finite representations, whereas the latter suggests conditions sufficient for the finiteness of the models.

The first paradigm hinges on the *finite unification* property, which is defined in the context of a backwards chaining procedure [16]. Informally, a set of rules exhibits this property if query answering can be performed via a finite rewriting of the initial query which can subsequently be evaluated over the initial set of facts. This provides a finite representation of the part of the model which is relevant to the query. Prominent classes of this paradigm include sticky rules [43, 45] and greedy bounded treewidth sets of rules [19]; it is decidable to recognise whether a set of rules belongs to these classes. The finite unification property follows by limitations on the form of the rules. For example, stickiness requires—among other things—that each rule that has a double occurrence of a certain variable in the body has at least one occurrence of the same variable in the rule head; e.g., rule r_3 of Example 5 violates stickiness.

The second paradigm relies on the *bounded treewidth* property [39] which characterises rule sets, such that the application of chase to them gives rise to a graph of bounded treewidth. Intuitively, the graph induced by chasing such sets of rules has a 'forest-like' shape. This category includes guarded rules [40] and generalisations thereof such as weakly guarded rules [39], weakly-frontier-guarded rules [16] and glut-frontier-guarded rules [149]. As before, these classes impose restrictions on the shape of the rules. For instance, guardedness requires that each rule contains a body

atom that covers all the universally quantified variables of the rule; thus rules such as r_7 of Example 5 are disallowed.

The final paradigm is related to the *finite expansion* (or *finite model*) property [18] and focuses on *acyclicity* conditions. Acyclicity conditions analyse the information flow between rules and try to detect cyclic applications of rules that may incur non-termination of the chase. Therefore, applying the chase to an acyclic set of rules is guaranteed to terminate. We already saw that conditions from the first two paradigms are overly restrictive for the rules of Example 5. Moreover, since we are interested in representing structures that are modelled up to a certain level of granularity and thus consist of a finite number of subparts, we choose the acyclicity paradigm as the most suitable for our setting. Next, we present an overview of conditions that are sufficient for chase termination and have been motivated by problems related to database dependencies, rule-based knowledge representation and logic programming.

3.3.1.1 Acyclicity for Databases

Acyclicity conditions were initially developed in the context of databases for the purposes of *data exchange* [80], where data structured under a source schema are transformed into data conforming to a target schema using existential rules. Another domain of usage for existential rules and databases is *ontology-based data access* (OBDA) [197], where classical extensional databases are enriched with an ontological layer consisting of existential rules in order to enable more sophisticated query answering. Existential rules in a database setting are usually referred to as *tuple-generating dependencies* (tgds); tgds form the foundation of Datalog$^{\pm}$ [42], a prominent formalism applications of which include data exchange and query answering over knowledge bases and databases.

Next, we present two chase termination conditions designed for database applications: *weak acyclicity* [80] and *super-weak acyclicity* [177]. After formally introducing them, we examine their applicability to Example 5.

Definition 7 (Weak Acyclicity)**.** *A* position *is an expression of the form* $\mathsf{p}|_\mathsf{i}$ *where* p *is an* n*-ary predicate and* $1 \leq i \leq n$*. For a set of rules* P*, the set* $\mathsf{Pos}(P)$ *contains each* $\mathsf{p}|_\mathsf{i}$

such that $\mathsf{p} \in \mathsf{Pred}(P)$ *and* $1 \leq \mathsf{i} \leq \mathsf{ar}(\mathsf{p})$. *For a variable* v *and a rule* $r = (B^+, \emptyset, H)$, *the set* $\mathsf{Pos}_B(\mathsf{v})$ *contains all the positions* $\mathsf{p}|_\mathsf{i}$ *such that* $\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{n}) \in B$ *and* $\mathsf{t}_\mathsf{i} = \mathsf{v}$; *the set* $\mathsf{Pos}_H(\mathsf{v})$ *contains all the positions* $\mathsf{p}|_\mathsf{i}$ *such that* $\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{n}) \in H$ *and* $\mathsf{t}_\mathsf{i} = \mathsf{v}$.

For a set of rules P, *the* weak acyclicity graph *of* P *(written* $\mathsf{WAG}(P)$*) is a directed graph whose set of vertices is* $\mathsf{Pos}(P)$ *and whose set of edges contains for each* $r \in P$, *for each* $\mathsf{x} \in \mathsf{fr}(r)$ *and for each existentially quantified variable* y *of* r

- *a* normal edge *from each position in* $\mathsf{Pos}_B(\mathsf{x})$ *to each position in* $\mathsf{Pos}_H(\mathsf{x})$ *and*
- *a* special edge *from each position in* $\mathsf{Pos}_B(\mathsf{x})$ *to each position in* $\mathsf{Pos}_H(\mathsf{y})$.

P *is* weakly acyclic (WA) *if* $\mathsf{WAG}(P)$ *does not contain a cycle going through a special edge.*

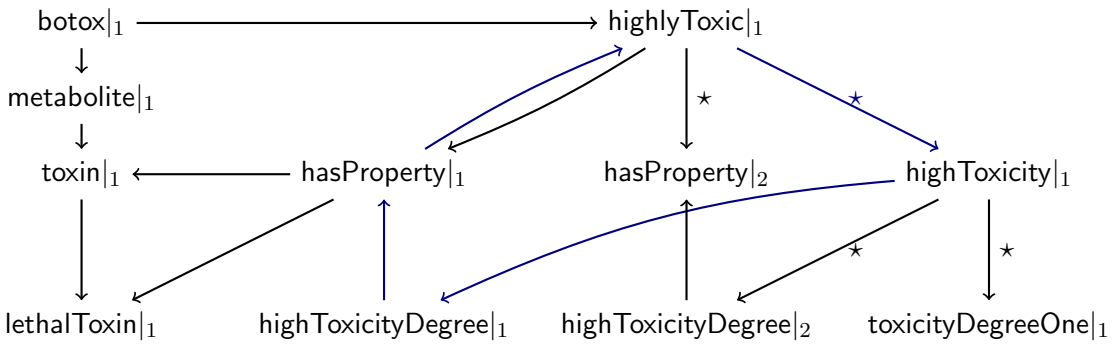


Figure 3.5: Weak acyclicity graph for program $P = \{r_i\}_{i=1}^{7}$ of Example 5

Figure 3.5 depicts $\mathsf{WAG}(P)$ for the set of rules $P = \{r_i\}_{i=1}^{7}$ of Example 5, where the symbol $\star$ is used to distinguish special edges from normal edges. One can observe from the edges of $\mathsf{WAG}(P)$ that P is not WA due to the existence of a cycle that goes through a special edge (from $\mathsf{highlyToxic}|_1$ to $\mathsf{highToxicity}|_1$). As a consequence, weak acyclicity fails to guarantee that chasing P over a set of facts is going to terminate. We next describe super-weak acyclicity that conducts a more fine-grained analysis of the propagation of values among the rules. Super-weak acyclicity is based on a more realistic estimate as to when one rule depends on another by requiring satisfaction of all body atoms that contain a frontier variable in the dependent rule and by performing unification checks.

Definition 8 (Super-weak acyclicity)**.** *A* place *is an expression of the form* $\langle \alpha, i \rangle$*, where* α *is an* n*-ary atom and* $1 \leq i \leq n$*. For a rule* $r = (B, \emptyset, H)$ *and a variable* v *of* r*, we define*

$$\mathsf{In}(r, \mathsf{v}) = \{\langle \mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n), \mathsf{i} \rangle \mid \mathsf{p}(\vec{\mathsf{t}}) \in B, \mathsf{t}_\mathsf{i} = \mathsf{v}\}$$

$$\mathsf{Out}(r, \mathsf{v}) = \{\langle \mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n)\theta_{\mathsf{sk}}, \mathsf{i} \rangle \mid \mathsf{p}(\vec{\mathsf{t}}) \in H, \mathsf{t}_\mathsf{i} = \mathsf{v}\}$$

where θ_{sk} *is the substitution such that* $r\theta_{\mathsf{sk}} = \mathsf{sk}(r)$*. For two sets of places* Q *and* Q'*, we have* $Q \sqsubseteq Q'$ *if for each* $\langle \alpha, i \rangle \in Q$ *there is* $\langle \alpha', i \rangle \in Q'$ *such that there exist substitutions* θ *and* θ' *for which* $\alpha\theta = \alpha\theta'$*.*

Let P *be a set of positive rules. For a rule* $r \in P$ *and an existentially quantified variable* y *of* r*, the set* $\mathsf{Move}_P(r, \mathsf{y})$ *is the minimal set such that*

- $\mathsf{Out}(r, \mathsf{y}) \subseteq \mathsf{Move}_P(r, \mathsf{y})$ *and*
- $\mathsf{Out}(r', \mathsf{v}) \subseteq \mathsf{Move}_P(r, \mathsf{y})$ *for each rule* $r' \in P$ *and each variable* $\mathsf{v} \in \mathsf{fr}(r)$ *with* $\mathsf{In}(r', \mathsf{v}) \sqsubseteq \mathsf{Move}_P(r, \mathsf{y})$*.*

For two rules $r, r' \in P$*, we have that* r triggers r' *under* P *(written* $r \rightsquigarrow_P r'$*) if there exists an existentially quantified variable* y *of* r *and a variable* $\mathsf{x} \in \mathsf{fr}(r')$ *such that* $\mathsf{In}(r', \mathsf{x}) \sqsubseteq \mathsf{Move}_P(r, \mathsf{y})$*.* P *is* super-weakly acyclic (SWA) *if the transitive closure of* $\rightsquigarrow_P$ *is irreflexive.*

Let us consider again program $P = \{r_i\}_{i=1}^{7}$ of Example 5. For rule r_2 we have

$$\begin{aligned}
\mathsf{In}(r_2, \mathsf{x}_2) &= \{\langle \mathsf{highlyToxic}(\mathsf{x}_2), 1 \rangle\} \\
\mathsf{Out}(r_2, \mathsf{y}_2) &= \{\langle \mathsf{hasProperty}(\mathsf{x}_2, \mathsf{y}_2), 2 \rangle, \langle \mathsf{highToxicity}(\mathsf{y}_2), 1 \rangle\} \\
\mathsf{Move}_P(r_2, \mathsf{y}_2) &= \mathsf{Out}(r_2, \mathsf{y}_2) \cup \{\langle \mathsf{hasToxicityDegree}(\mathsf{x}_4, \mathsf{y}_4), 1 \rangle, \\
&\qquad \langle \mathsf{hasProperty}(\mathsf{x}_5, \mathsf{w}_5), 1 \rangle, \langle \mathsf{highlyToxic}(\mathsf{x}_3), 1 \rangle\}
\end{aligned}$$

For $\theta = \{\mathsf{x}_2 \mapsto \mathsf{b}\}$ and $\theta' = \{\mathsf{x}_3 \mapsto \mathsf{b}\}$ we have

$$\mathsf{highlyToxic}(\mathsf{x}_2)\theta = \mathsf{highlyToxic}(\mathsf{x}_3)\theta'$$

from which we derive $\mathsf{In}(r_2, \mathsf{x}_2) \sqsubseteq \mathsf{Move}_P(r_2, \mathsf{y}_2)$; thus, $r_2 \rightsquigarrow_P r_2$ and P is not SWA.

Super-weak acyclicity strictly captures weak acyclicity in terms of expressive power [178] and both conditions can be checked in polynomial time. In the same spirit, *safety* [180] is a polynomially checkable condition that inspects only a subgraph of the weak acyclicity graph for cycles, where the subgraph consists only of *affected positions*; affected positions were first introduced by Calì et al. in order to single out positions where function terms may occur during the chase run [40]. Another chase termination condition that draws upon weak acyclicity is $\prec$-*stratification* [71], where the program is split in subsets and each subset is checked for weak acyclicity; testing $\prec$-stratification is coNP-complete. A different approach proposes checking chase termination by rewriting the set of rules and then applying any of the known acyclicity condition to the rewritten program; however, no complexity bounds are provided for these conditions [221].

3.3.1.2 Acyclicity for Rule-Based Knowledge Representation

A number of acyclicity criteria have also emerged from the field of rule-based knowledge representation. We next present *joint acyclicity*, a condition sufficient for chase termination that has been motivated by information integration applications [149].

Definition 9 (Joint Acyclicity)**.** *For a set of positive rules P, a rule $r \in P$ and an existentially quantified variable y in r, the set $\mathsf{jMove}_P(r, \mathsf{y})$ is the minimal subset of $\mathsf{Pos}(P)$ such that*

- $\mathsf{Pos}_H(\mathsf{y}) \subseteq \mathsf{jMove}_P(r, \mathsf{y})$ *and*
- $\mathsf{Pos}_H(\mathsf{x}) \subseteq \mathsf{jMove}_P(r, \mathsf{y})$ *for each $r \in P$ and for each $\mathsf{x} \in \mathsf{fr}(r)$ such that $\mathsf{Pos}_B(\mathsf{x}) \subseteq \mathsf{jMove}_P(r, \mathsf{y})$.*

Let P be a set of rules. The joint acyclicity graph *of P (written $\mathsf{JAG}(P)$) is a directed graph that contains the existentially quantified variables in P as its vertices and also contains an edge from an existentially quantified variable y_1 of a rule r_1 to an existentially quantified variable y_2 of a rule r_2 if there exists $\mathsf{x}_2 \in \mathsf{fr}(r_2)$ such that $\mathsf{Pos}_B(\mathsf{x}_2) \subseteq \mathsf{jMove}_P(r_1, \mathsf{y}_1)$. P is* jointly acyclic (JA) *if $\mathsf{JAG}(P)$ does not contain a directed cycle.*

One can check that for the program $P = \{r_i\}_{i=1}^{7}$ of Example 5, $\mathsf{JAG}(P)$ contains a self-edge from y_2 to y_2 because of $\mathsf{Pos}_B(\mathsf{x}_2) = \{\mathsf{highlyToxic}|_1\}$ and $\mathsf{jMove}_P(r_2, \mathsf{y}_2) = \{\mathsf{hasProperty}|_2, \mathsf{highToxicity}|_1, \mathsf{hasToxicityDegree}|_1, \mathsf{hasProperty}|_1, \mathsf{highlyToxic}|_1\}$; so, P is not JA.

Other approaches for testing finiteness of models include identifying *dependencies* between rules, that is examining whether it is possible for one rule to trigger another; one can easily guarantee termination of the chase for sets of rules such that for each rule no chain of dependencies is observed from a rule to itself. Baget et al. have formulated such a chase termination criterion that draws upon dependencies and is known as *acyclic graph of dependencies (aGRD)* [17]. Baget et al. have also suggested refinements of aGRD that capture strictly more sets of rules by taking into account whether a rule triggers another one after repeatedly applying chase for a fixed number of steps k; such dependencies were coined as 'farsighted dependencies' [17, 20]. We will discuss more in depth this group of dependency-based acyclicity conditions in Chapter 4.

3.3.1.3 Acyclicity for Logic Programs

Acyclicity conditions were formulated in the logic programming setting in order to identify logic programs with function symbols whose grounding is finite. Since extensions of logic programming languages support a variety of additional features such as nonmonotonic negation, disjunction and arbitrary use of function symbols, the corresponding acyclicity conditions were designed to deal with these features. In this section, we recapitulate the definition of *finite domain* condition, which is a decidable condition for recognising logic programs with stable models of finite size; Calimeri et al. [47] describe an algorithm which checks the finite domain condition in polynomial time. Since we are only interested in logic programs with nonmonotonic negation and function symbols that are a result of skolemising existential rules, we suitably simplify the definition of finite domain condition and adjust it to our setting.

Definition 10 (Finite Domain Condition)**.** *Let P be a set of rules; moreover, let $P_+ = \{(B^+, \emptyset, H) \mid (B^+, B^-, H) \in P\}$. A position $\mathsf{p}|_\mathsf{i}$ is P-*recursive *with another*

position $\mathsf{q}|_\mathsf{j}$ *if* $\mathsf{WAG}(P_+)$ *contains a directed cycle (consisting of normal and/or special edges) that goes through* $\mathsf{p}|_\mathsf{i}$ *and* $\mathsf{q}|_\mathsf{j}$. *The set of* finite domain positions *of* P *(written* $\mathsf{Pos}_{FD}(P)$*) is the maximal subset of* $\mathsf{Pos}(P)$ *such that for each* $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_{FD}(P)$, *for each rule* r, *for each atom* $\mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{n})$ *occurring in the head of* r *and for* $i \in \{1, \ldots, n\}$

- *if* $\mathsf{t}_\mathsf{i} \in \mathsf{fr}(r)$, *then* $\mathsf{Pos}_B(\mathsf{t}_\mathsf{i}) \cap \mathsf{Pos}_{FD}(P) \neq \emptyset$ *and*
- *if* t_i *is an existentially quantified variable, then for each* $\mathsf{x} \in \mathsf{fr}(r)$ *there is* $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_B(\mathsf{x}) \cap \mathsf{Pos}_{FD}(P)$ *such that* $\mathsf{q}|_\mathsf{j}$ *is not* P*-recursive with* $\mathsf{p}|_\mathsf{i}$.

P *is* finite domain (FD) *if* $\mathsf{Pos}_{FD}(P) = \mathsf{Pos}(P)$.

Let us now examine the set of finite domain positions for program $P = \{r_i\}_{i=1}^{7}$ of Example 5. We have $\mathsf{botox}|_1 \in \mathsf{Pos}_{FD}(P)$ because the predicate botox does not occur in the head of any rule. However, one can observe that $\mathsf{hasProperty}|_1 \notin \mathsf{Pos}_{FD}(P)$ because in rule r_2 we have $\mathsf{Pos}_B(\mathsf{x}_2) = \{\mathsf{highlyToxic}|_1\}$ and $\mathsf{hasProperty}|_1$ is P-recursive with $\mathsf{highlyToxic}|_1$ as one can infer from the cycle that goes through $\mathsf{highlyToxic}|_1$ and $\mathsf{hasProperty}|_1$ in Figure 3.5. Therefore, P is not FD.

We next discuss *argument-restrictedness*, which is another acyclicity condition formulated for logic programs, which strictly subsumes FD [157]. Lierler and Lifschtiz prove that argument-restrictedness guarantees finiteness of the stable model and can be checked in polynomial time.

Definition 11 (Argument-restrictedness)**.** *Let* P *be a set of rules; moreover, let* $P_+ = \{(B^+, \emptyset, H) \mid (B^+, B^-, H) \in P\}$. *An* argument ranking *is a mapping* α *from* $\mathsf{Pos}(P)$ *to the set of nonnegative integers such that the following conditions are satisfied for each rule* $r \in P$, *each variable* $\mathsf{x} \in \mathsf{fr}(r)$ *and each existentially quantified variable* y *in* r*:*

1. *for each* $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{x})$, *some* $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_B(\mathsf{x})$ *exists such that* $\alpha(\mathsf{p}|_\mathsf{i}) \geq \alpha(\mathsf{q}|_\mathsf{j})$ *and*
2. *for each* $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{y})$, *some* $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_B(\mathsf{x})$ *exists such that* $\alpha(\mathsf{p}|_\mathsf{i}) > \alpha(\mathsf{q}|_\mathsf{j})$.

P *is* argument restricted *(AR) if an argument ranking for* P *exists.*

We next show that P is not AR. Assume for brevity that $n_1 = \alpha(\mathsf{highToxicity}|_1)$, $n_2 = \alpha(\mathsf{hasProperty}|_1)$, $n_3 = \alpha(\mathsf{highToxicity}|_1)$ and $n_4 = \alpha(\mathsf{hasToxicityDegree}|_1)$. By r_2 and r_3, we have $n_2 \geq n_1$ and $n_1 \geq n_2$ that imply $n_1 = n_2$. By r_2, r_4 and r_5, we have respectively $n_3 > n_1$, $n_4 \geq n_3$ and $n_2 \geq n_4$, which imply $n_2 > n_1$. The latter in conjunction with $n_1 = n_2$ results in a contradiction.

The treatment of functional terms has received much attention from logic programming researchers due to the direct applicability of finite grounding conditions to the implementation of logic programming engines [215, 67]. Further such conditions that have been proposed include ω-restrictedness [228] and λ-restrictedness [227] but without any complexity considerations for checking them (the articles that introduced ω-restrictedness and λ-restrictedness do not mention whether checking them is decidable). The conditions of ω-restrictedness, λ-restrictedness and argument-restrictedness have been incorporated in the SMODELS [227], GRINGO [89] and DLV [153] answer set systems, respectively. A more recent acyclicity condition for logic programs with function symbols is Γ-acyclicity [103] which involves a check similar to the one used for $\prec$-stratification discussed above; Γ-acyclicity is proved to be decidable but no complexity estimation is provided for it. However, given that $\prec$-stratification is coNP-hard, verifying Γ-acyclic programs should be no easier than that.

Regarding the expressive power of these conditions Gebser et al. [89] prove that λ-restrictedness subsumes ω-restrictedness, whereas Calimeri et al. [47] show that the finite domain condition strictly generalises λ-restrictedness; Lin and Wang [159] also suggest an answer set programming formalism with function symbols but their language is strictly contained in the class of ω-restricted programs. On the other hand, Lierler and Lifschitz [157] demonstrate that argument-restrictedness strictly captures both λ-restrictedness and (as we already saw) the finite domain condition. Γ-acyclicity generalises the finite domain condition, but neither does Γ-acyclicity contain argument-restrictedness nor does argument-restrictedness contain Γ-acyclicity [100]. In Section 4.3 of Chapter 4, we will provide a detailed technical comparison between the various acyclicity conditions originating from all three areas.

3.3.2 Limitations Imposed by Stratified Negation-as-Failure

In this work, we are interested in logic programs with at most one stable model. Uniqueness of the stable model is a desirable property, because it implies that reasoning tasks can be carried out using a deterministic procedure. A class of logic programs that often have more than one stable model is the class of disjunctive logic programs [75]. Disjunctive logic programs are extensions of logic programs such that disjunctions of atoms are allowed in the head of the rules and such that the stable model semantics have been appropriately modified to take into account the disjunctions [201]. The use of disjunction permits to encode situations where more than one scenario needs to be considered in parallel. In this thesis, we focus on the modelling of complex objects with a uniquely determined structure and we, thus, do not consider disjunction. Another source of non-determinism that may result in more than one stable model is the recursive use of nonmonotonic negation. For instance, consider the logic program $P = \{\mathsf{a}(0), r_{14}, r_{15}\}$, where r_{14} and r_{15} are defined as follows.

$$\mathsf{a}(\mathsf{x}) \wedge \ \textbf{not}\ \mathsf{b}(\mathsf{x}) \rightarrow \mathsf{c}(\mathsf{x}) \qquad (r_{14})$$

$$\mathsf{a}(\mathsf{x}) \wedge \ \textbf{not}\ \mathsf{c}(\mathsf{x}) \rightarrow \mathsf{b}(\mathsf{x}) \qquad (r_{15})$$

P has exactly two stable models, namely $\mathcal{M}_1 = \{\mathsf{a}(0), \mathsf{b}(0)\}$ and $\mathcal{M}_2 = \{\mathsf{a}(0), \mathsf{c}(0)\}$. Nevertheless, in all the other examples discussed so far, we extensively used negation within programs with exactly one stable model. This is because every logic program that we presented exhibits an important property that guarantees existence of at most one stable model at the presence of negation. This is the well-known property of stratification, which we formally introduce next [8].

Definition 12 (Stratification)**.** *Let P be a logic program. A* stratification *of P is a sequence of disjoint programs $\vec{\mathsf{P}} = P_1, \ldots, P_n$ such that $\bigcup_{1 \leq i \leq n} P_i = P$ and for all programs $P_i, P_j \in \vec{\mathsf{P}}$, rules $(B_1^+, B_1^-, H_1) \in P_i$ and $(B_2^+, B_2^-, H_2) \in P_j$, and every predicate $\mathsf{R} \in \mathsf{Pred}(H_1)$, we have:*

- *if $\mathsf{R} \in \mathsf{Pred}(B_2^+)$, then $i \leq j$ and*
- *if $\mathsf{R} \in \mathsf{Pred}(B_2^-)$, then $i < j$.*

The elements of $\vec{\mathsf{P}}$ *are called* strata. *P is* stratified *if it has a stratification.*

As pointed out above, stratified programs are satisfied by at most one stable model, the computation of which relies on the stratification of the programs and the use of the *consequence operator* which we introduce next [9].

Definition 13 (Consequence Operator)**.** *For a rule $r \in P$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a set of facts F, the* application *of r to F, written $r(F)$, is defined as*

$$r(F) = \bigcup_{B^+\theta \subseteq F,\ B^-\theta \cap F = \emptyset} H\theta$$

where θ is a substitution from $\mathsf{Var}(B^+)$ to $\mathsf{Const}(F)$. Also, let $T_P(F) = F \cup \bigcup_{r \in P} r(F)$ and define $T_P^0(F) = F$, $T_P^{i+1}(F) = T_P(T_P^i(F))$ and $T_P^\infty(F) = \bigcup_{i \geq 0} T_P^i(F)$.

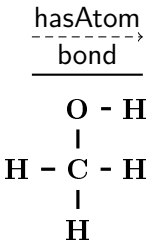
Please note that Definition 13 supersedes Definition 4 for the application of r to F which makes $r(F)$ uniquely defined and does not cause any notation conflict. As we see next, the T_P operator can be used to characterise stable models [9].

Proposition 14. *For a program P, a set of facts F and a stable model $\mathcal{M} \models_{\mathsf{SM}} P \cup F$, we have $\mathcal{M} = T^\infty_{\mathsf{GL}(P,\mathcal{M})}(F)$.*

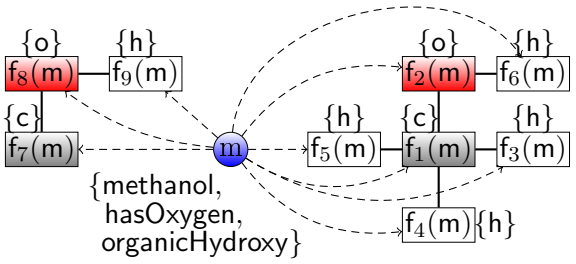
For stratified programs the T_P operator provides us with a deterministic computation procedure [9].

Proposition 15. *Let P be a program P and let F be a set of facts; if $\vec{\mathsf{P}} = P_1, \ldots, P_n$ is a stratification of P, then $\mathcal{M} = T^\infty_{P_n}(\ldots T^\infty_{P_1}(F)\ldots)$ is the unique stable model of P if $\bot \notin \mathcal{M}$; otherwise P has no stable model.*

Therefore, stratification is a sufficient property for uniqueness of the stable model, that allows us to employ a deterministic algorithm for the standard reasoning tasks. Note that programs that consist entirely of positive rules are stratified by default and therefore have at most one stable model too. However, stratification is not a necessary condition to guarantee determinism of the computation, that is stratified programs are strictly contained in the class of logic programs that have at most one stable model for every set of facts. Next, we present Example 16 which highlights

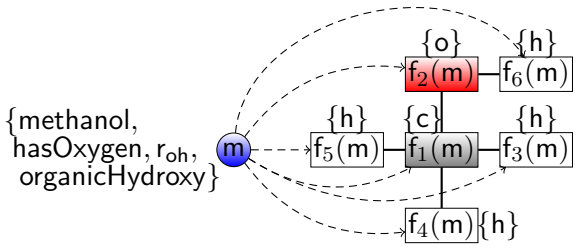


(a) Methanol molecule



(c) Representation of methanol by $P_m \cup P_h$

(b) Organic hydroxy part



(d) Representation of methanol by P

Figure 3.6: Chemical structures and models of Example 16

a case where stratification fails to characterise programs with a unique stable model whose rules are essential for the representation of structured objects.

Example 16. As we have already seen, structured objects can be modelled with nonmonotonic existential rules for the purposes of automatic logic-based classification. Similarly to the molecules of benzene and cyclobutane, one can represent the molecular structure of methanol (see Figure 3.6(a)) using rule g_m.

$$\mathsf{methanol}(\mathsf{x}_1) \rightarrow \exists_{\mathsf{i}=1}^{6}\mathsf{y}_\mathsf{i}.\mathsf{molecule}(\mathsf{x}_1) \wedge \bigwedge_{\mathsf{i}=1}^{6}\mathsf{hasAtom}(\mathsf{x}_1,\mathsf{y}_\mathsf{i}) \wedge \mathsf{carbon}(\mathsf{y}_1) \wedge \mathsf{oxygen}(\mathsf{y}_2) \wedge \bigwedge_{\mathsf{i}=3}^{6}\mathsf{hydrogen}(\mathsf{y}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=2}^{5}\mathsf{singleBond}(\mathsf{y}_1,\mathsf{y}_\mathsf{i}) \wedge \mathsf{singleBond}(\mathsf{y}_2,\mathsf{y}_6) \qquad (g_m)$$

Moreover, one can recognise organic hydroxy chemicals (i.e. structures with the C-O-H subpart as depicted by Figure 3.6(b)) with r_1, structures that contain oxygen with r_{ho} and molecules that contain exactly one or more than one carbon atoms with r_{oc}

and r_{mc}, respectively.

$$\bigwedge_{i=1}^{3} \mathsf{hasAtom}(u, v_i) \wedge \mathsf{carbon}(v_1) \wedge \mathsf{oxygen}(v_2) \wedge \mathsf{hydrogen}(v_3) \wedge \mathsf{singleBond}(v_1, v_2) \wedge \mathsf{singleBond}(v_2, v_3) \rightarrow \mathsf{organicHydroxy}(u) \quad (r_1)$$

$$\mathsf{hasAtom}(x, z) \wedge \mathsf{oxygen}(z) \rightarrow \mathsf{hasOxygen}(x) \quad (r_{ho})$$

$$\mathsf{molecule}(x) \wedge \mathsf{hasAtom}(x, z_1) \wedge \mathsf{carbon}(z_1) \wedge \mathsf{hasAtom}(x, z_2) \wedge \mathsf{carbon}(z_2) \wedge z_1 \neq z_2 \rightarrow \mathsf{multiCarbonMolecule}(x) \quad (r_{mc})$$

$$\mathsf{molecule}(x) \wedge \mathsf{hasAtom}(x, z) \wedge \mathsf{carbon}(z) \wedge \mathbf{not}\ \mathsf{multiCarbonMolecule}(x) \rightarrow \mathsf{oneCarbonMolecule}(x) \quad (r_{oc})$$

By computing the stable model $\mathcal{M}_m$ of $P_m = \{\mathsf{sk}(g_m), r_1, r_{ho}, r_{oc}, r_{mc}\}$ when extended with $\mathsf{methanol}(m)$ one can classify methanol.

$$\mathcal{M}_m = \{\mathsf{methanol}(m), \mathsf{carbon}(f_1(m)), \mathsf{oxygen}(f_2(m))\} \cup \{\mathsf{hasAtom}(m, f_i(m)\}_{i=1}^{6} \cup \{\mathsf{hydrogen}(f_i(m))\}_{i=3}^{6} \cup \{\mathsf{singleBond}(f_1(m), f_i(m))\}_{i=2}^{5} \cup \{\mathsf{singleBond}(f_2(m), f_6(m))\} \cup \{\mathsf{molecule}(m), \mathsf{organicHydroxy}(m), \mathsf{hasOxygen}(m), \mathsf{oneCarbonMolecule}(m)\}$$

By the atoms contained in $\mathcal{M}_m$, one can infer that

$$P_m \models \mathsf{methanol} \sqsubseteq \mathsf{molecule}$$

$$P_m \models \mathsf{methanol} \sqsubseteq \mathsf{hasOxygen}$$

$$P_m \models \mathsf{methanol} \sqsubseteq \mathsf{organicHydroxy}$$

$$P_m \models \mathsf{methanol} \sqsubseteq \mathsf{oneCarbonMolecule}$$

However, one is usually interested not only in the subclass relations between molecules and chemical classes (two-level hierarchy) but also in subsumptions between the chemical classes themselves. So, in order to determine the subsumers of organic hydroxy molecules, one needs to use a rule such as r_2 to describe their structure.

$$\mathsf{organicHydroxy}(x_2) \rightarrow \exists_{i=7}^{9} y_i . \bigwedge_{i=7}^{9} \mathsf{hasAtom}(x_2, y_i) \wedge \mathsf{carbon}(y_7) \wedge \mathsf{oxygen}(y_8) \wedge \mathsf{hydrogen}(y_9) \wedge \mathsf{singleBond}(y_7, y_8) \wedge \mathsf{singleBond}(y_8, y_9) \quad (r_2)$$

As usual, one can classify organic hydroxy using the stable model $\mathcal{M}_h$ of the program $P_h = \{\mathsf{sk}(r_2), r_{ho}, r_{oc}, r_{mc}\}$ extended with the fact $\mathsf{organicHydroxy(h)}$.

$$\begin{aligned}\mathcal{M}_h = &\{\mathsf{organicHydroxy(h)}\} \cup \{\mathsf{hasAtom(h, f_i(h))}\}_{\mathsf{i}=7}^{8} \cup \{\mathsf{carbon(f_7(h)), oxygen(f_8(h))}\} \cup \\ &\{\mathsf{hydrogen(f_9(h)), singleBond(f_7(h), f_8(h)), singleBond(f_8(h), f_9(h)), hasOxygen(h)}\}\end{aligned}$$

We can thus deduce that $P_h \models \mathsf{organicHydroxy} \sqsubseteq \mathsf{hasOxygen}$. Nevertheless, in case we want to uncover the subsumers of both methanol and organic hydroxy with the same program the stable model of $P_m \cup P_h \cup \{\mathsf{methanol(m), organicHydroxy(h)}\}$ is not $\mathcal{M}_m \cup \mathcal{M}_h$, but $\mathcal{M}'$ instead.

$$\begin{aligned}\mathcal{M}' = &(\mathcal{M}_m \setminus \{\mathsf{oneCarbonMolecule(m)}\}) \cup \mathcal{M}_h \cup \{\mathsf{hasAtom(m, f_i(m)}\}_{\mathsf{i}=7}^{9} \cup \\ &\{\mathsf{carbon(f_7(m)), oxygen(f_8(m)), hydrogen(f_9(m))}\} \cup \\ &\{\mathsf{singleBond(f_7(m), f_8(m)), singleBond(f_8(m), f_9(m)), multiCarbonMolecule(m)}\}\end{aligned}$$

As a consequence, we have $P_m \cup P_h \not\models \mathsf{methanol} \sqsubseteq \mathsf{oneCarbonMolecule}$ and also that $P_m \cup P_h \models \mathsf{methanol} \sqsubseteq \mathsf{multiCarbonMolecule}$ since m contains an additional carbon in $\mathcal{M}'$. This is a consequence of the fact that two instantiations of r_2 have to be satisfied: one for generating the structure of organic hydroxy h (rule body $\mathsf{organicHydroxy(h)}$) and once for generating the C-O-H part of methanol m (rule body $\mathsf{organicHydroxy(m)}$) that has been recognised as an organic hydroxy. So, methanol is no longer faithfully represented as it contains extra atoms, which incurs missing ('methanol is an one-carbon molecule') and wrong ('methanol is a multi-carbon molecule') subsumptions. In order to overcome this problem, one can insert suitable auxiliary atoms in r_1 and r_2 which will ensure that the stable model of the knowledge base is the intended one. Rules r_1 and r_2 can be rewritten into r_{oh} and g_{oh} as follows.

$$\begin{aligned}&\bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom(u, v_i)} \wedge \mathsf{carbon(v_1)} \wedge \\ &\qquad \mathsf{oxygen(v_2)} \wedge \mathsf{hydrogen(v_3)} \wedge \\ &\mathsf{singleBond(v_1, v_2)} \wedge \mathsf{singleBond(v_2, v_3)} \wedge \\ &\mathbf{not}\ \mathsf{g_{oh}(v_1)} \wedge\ \mathbf{not}\ \mathsf{g_{oh}(v_2)} \wedge\ \mathbf{not}\ \mathsf{g_{oh}(v_3)} \rightarrow \mathsf{organicHydroxy(u)} \wedge \mathsf{r_{oh}(u)} \qquad (r_{oh})\end{aligned}$$

$$\mathsf{organicHydroxy}(\mathsf{x}_2) \wedge \ \mathbf{not}\ \mathsf{r_{oh}}(\mathsf{x}_2) \rightarrow \exists_{\mathsf{i}=7}^{9}\mathsf{y_i}.\ \bigwedge_{\mathsf{i}=7}^{9} \mathsf{hasAtom}(\mathsf{x}_2, \mathsf{y_i}) \wedge \mathsf{carbon}(\mathsf{y}_7) \wedge$$
$$\mathsf{oxygen}(\mathsf{y}_8) \wedge \mathsf{hydrogen}(\mathsf{y}_9) \wedge$$
$$\mathsf{singleBond}(\mathsf{y}_7, \mathsf{y}_8) \wedge \mathsf{singleBond}(\mathsf{y}_8, \mathsf{y}_9) \wedge$$
$$\mathsf{g_{oh}}(\mathsf{y}_7) \wedge \mathsf{g_{oh}}(\mathsf{y}_8) \wedge \mathsf{g_{oh}}(\mathsf{y}_9) \qquad (g_{oh})$$

The program $P = \{\mathsf{sk}(g_m), r_{oh}, \mathsf{sk}(g_{oh}), r_{ho}, r_{oc}, r_{mc}\}$ extended with $\mathsf{methanol}(\mathsf{m})$ and $\mathsf{organicHydroxy}(\mathsf{h})$ has $\mathcal{M}_m \cup \mathcal{M}_h \cup \{\mathsf{g_{oh}}(\mathsf{f}_7(\mathsf{h})), \mathsf{g_{oh}}(\mathsf{f}_8(\mathsf{h})), \mathsf{g_{oh}}(\mathsf{f}_9(\mathsf{h})), \mathsf{r_{oh}}(\mathsf{m})\}$ as its only stable model, from which all desired subsumptions follow. Although P has a unique stable model it is obviously not stratified (due to the use of predicates $\mathsf{g_{oh}}$ and $\mathsf{r_{oh}}$) which means that there exists no stratification of P that allows deterministic computation of the stable model. ♢

Example 16 demonstrates the need for a more fine-grained notion of stratification that goes beyond disallowing recursive use of negation and is able to identify a broader class of programs for which deterministic reasoning procedures exist.

3.3.3 Intricate Syntax for Application Experts

The design of knowledge bases using rules such as the ones introduced in Section 3.2 can be counterintuitive for knowledge engineers that are not familiar with them. Graphical languages instead have been somewhat easier to use (albeit they do present problems such as lack of reading order and scalability) and thus more popular for describing scientific knowledge. This is witnessed by the wealth of diagrammatic languages adopted by life scientists, such as chemical graphs [230] or biological graphical notation [152]. For an ontology language, a human-friendly syntax that is based on a few keywords and can produce text-based legible expressions can be of help to ontology engineers. For instance, the following expression models the 'hydrocarbon' concept in a more natural way than rules r_{10} and r_{11} (page 45) do.

hydrocarbon SuperClassOf

carbonMolecule AND hasAtom ONLY (carbon OR hydrogen)

As similar problems had been encountered by ontology developers when modelling knowledge with OWL, Horridge et al. designed the Manchester OWL Syntax—a 'less logician-like' syntax for creating OWL axioms that was well received by application experts [123]. Similar efforts have been made for extensions of OWL with rules and with FOL Description Graphs [94]. In the same vein, a surface syntax for our setting that is less complicated than logic programming rules, but closer to natural language and uniquely translatable to logical axioms could enhance the usability of nonmonotonic existential rules by ontology developers.

Chapter 4

Stable Model Finiteness

In this chapter we present and analyse a number of sufficient conditions that ensure finiteness of the stable model(s) for a logic program. Initially, we outline conditions for programs that consist exclusively of positive rules; these conditions can also be viewed as chase termination criteria (published in [99]). Subsequently, we shift towards programs that may contain nonmonotonic negation and introduce new conditions for them (published in [170]). We conclude the chapter by analysing the relationship between the newly suggested and previously known termination conditions originating from areas such as databases, logic programming and rule-based knowledge representation (published in [100]).

Section 4.1 focuses on chase termination criteria for positive existential rules, whereas in Section 4.2 we turn our attention to acyclicity criteria for rules that also contain negative bodies. Section 4.3 contains discussion and comparison across a range of acyclicity conditions.

4.1 Acyclicity Conditions for Positive Existential Rules

We begin our presentation by introducing two novel acyclicity conditions for negation-free programs. In spite of the initial motivation being representation of complex objects, our results are of a wider interest and can be exploited for a variety of applications such as data exchange, logic programming and query answering over

description logic ontologies.

Initially, in Section 4.1.1 we formulate a general but expensive to check acyclicity condition; in Section 4.1.2 we sacrifice some generality in order to define a more tractable condition. Thereafter, in Section 4.1.3 we prove upper and lower complexity bounds for reasoning over acyclic programs of different kinds.

4.1.1 Model Faithful Acyclicity

The acyclicity conditions we presented in Section 3.3.1 attempt to determine whether an application of a rule may generate facts that can repeatedly (directly or indirectly) trigger the same rule infinitely many times. The main difference between the various criteria is how rule applicability is estimated. The majority of these conditions consider each input variable of a rule in isolation and thus do not check satisfaction of all body atoms at once; as a consequence, the existing conditions fail to detect chase termination for many programs, such as program $P = \{r_i\}_{i=1}^{7}$ from Example 5.

In order to obtain more precise chase termination guarantees, one needs to trace rule applicability more faithfully. This can be attained by exploiting the execution of the skolem chase so as to dynamically identify cyclic computations. Since it is not straightforward to determine what constitutes a 'cyclic computation', we treat an execution as cyclic when a term of the form $\mathsf{f}(\vec{\mathsf{t}})$, where f occurs in $\vec{\mathsf{t}}$, is derived. This condition can be further refined, e.g. by considering whether f occurs nested in a term some fixed number of times. In the current work, we only consider the case where f appears nested in a term at least once; it would be interesting for future work to relax this restriction by allowing f to occur nested within a term more than once. Next, we provide the definition of a cyclic term, which will be used later to characterise programs as acyclic if and only if their chase does not contain such a term.

Definition 17 (Cyclic term)**.** *A term* t *is* cyclic *if it has a subterm* $\mathsf{f}(\vec{\mathsf{t}})$ *that has a proper subterm of the form* $\mathsf{f}(\vec{\mathsf{s}})$*.*

We next present a novel declarative notion of acyclicity that tracks rule applicability by recording rule application. In order to check whether a set of rules P is acyclic,

P is converted into a new set of rules P' where fresh predicates are used to mark newly generated terms and the subterm relation between them. We define a program P to be acyclic if the transformed version of P does not entail a special propositional symbol Cycle. Since acyclicity is defined via entailment, it is not algorithm-specific and can be decided using any theorem proving procedure for existential rules that is sound and complete. We adopt for our condition the name *model-faithful acyclicity* (MFA) because it estimates rule application rigorously, by examining the actual structure of the minimal Herbrand model of P.

Before proceeding, let us provide some preliminary notation that we omitted from Chapter 2 so that it appears closer to where it is used for the first time. An *extended substitution* is a mapping from a set of terms to another set of terms; we define application of an extended substitution to an atom and a set of atoms in a way similar to application of a substitution. For two sets of atoms A and A', an *extended homomorphism* from A to A' is an extended substitution $\sigma : \mathsf{Terms}(A) \to \mathsf{Terms}(A')$, such that $A\sigma \subseteq A'$. Moreover, a *Boolean conjunctive query (BCQ)* is a formula of the form $\exists \vec{\mathsf{y}}.\phi(\vec{\mathsf{y}})$, where $\phi(\vec{\mathsf{y}})$ is a conjunction of atoms and each variable that occurs in $\phi(\vec{\mathsf{y}})$ is in $\vec{\mathsf{y}}$. For a set of rules P and a set of facts F, we have $F \cup P \models \exists \vec{\mathsf{y}}.\phi(\vec{\mathsf{y}})$ if there exists a homomorphism from $\phi(\vec{\mathsf{y}})$ to $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$. For each n-ary predicate P, let $\hat{\mathsf{P}}$ be a fresh $n+1$-ary predicate unique for P. For each term t and set of atoms R, let $\mathsf{Aster}(\mathsf{t}, R) = \{\hat{\mathsf{P}}(\mathsf{t}, \vec{\mathsf{t}}) \mid \mathsf{P}(\vec{\mathsf{t}}) \in R, |\vec{\mathsf{t}}| \geq 0\}$. Finally, for a set of rules P and a variable w let $\mathsf{Aster}(P)$ be the smallest set of rules containing $\mathsf{Aster}(\mathsf{w}, B) \to \mathsf{Aster}(\mathsf{w}, H)$ for each rule $B \to H$ in P.

For the remainder of Section 4.1, we denote each rule r of the form (2.1) with $\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}}) \to \exists \mathsf{y}.\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{y}})$, such that $\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}})$ and $\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{y}})$ are conjunctions of atoms; for the time being we restrict our discussion to positive rules, i.e. in the rest of this section when we mention a rule we imply a positive rule.

Definition 18. *For each rule* $r = \mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}}) \to \exists \vec{\mathsf{y}}.\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{y}})$ *and each variable* $\mathsf{y}_\mathsf{i} \in \vec{\mathsf{y}}$*, let* N^i_r *be a fresh unary predicate unique for* r *and* y*, let* Succ *and* Desc *be fresh binary*

predicates and let Cycle *be a fresh nullary predicate.* $\mathsf{MFA}(r)$ *is defined as follows:*

$$\mathsf{B}(\vec{\mathsf{x}},\vec{\mathsf{z}}) \rightarrow \exists \vec{\mathsf{y}}.\left[\mathsf{H}(\vec{\mathsf{x}},\vec{\mathsf{y}}) \wedge \bigwedge_{\mathsf{y_i}\in\vec{\mathsf{y}}}\left[\mathsf{N}^{\mathsf{i}}_{r}(\mathsf{y_i}) \wedge \bigwedge_{\mathsf{x_j}\in\vec{\mathsf{x}}} \mathsf{Succ}(\mathsf{x_j},\mathsf{y_i})\right]\right]$$

For a set of rules P, $\mathsf{MFA}(P)$ *is the smallest set that contains* $\mathsf{MFA}(r)$ *for each rule* $r \in P$, *rule* (4.1), *rule* (4.2) *and rule* (4.3) *instantiated for each* $\mathsf{N}^{\mathsf{i}}_{r}$, *such that* $\mathsf{y_i}$ *is the* i*-th existentially quantified variable occurring in the head of some rule* r*:*

$$\mathsf{Succ}(\mathsf{x}_1,\mathsf{x}_2) \rightarrow \mathsf{Desc}(\mathsf{x}_1,\mathsf{x}_2) \tag{4.1}$$

$$\mathsf{Desc}(\mathsf{x}_1,\mathsf{x}_2) \wedge \mathsf{Succ}(\mathsf{x}_2,\mathsf{x}_3) \rightarrow \mathsf{Desc}(\mathsf{x}_1,\mathsf{x}_3) \tag{4.2}$$

$$\mathsf{N}^{\mathsf{i}}_{r}(\mathsf{x}_1) \wedge \mathsf{Desc}(\mathsf{x}_1,\mathsf{x}_2) \wedge \mathsf{N}^{\mathsf{i}}_{r}(\mathsf{x}_2) \rightarrow \mathsf{Cycle} \tag{4.3}$$

P *is* model-faithful acyclic *(MFA) w.r.t. a set of facts* F *if* $F \cup \mathsf{MFA}(P) \not\models \mathsf{Cycle}$. *The* critical instance I^*_P *is the set of facts that can be constructed using* $\mathsf{Pred}(P)$, $\mathsf{Const}(P)$ *and a special fresh constant* $*$; P *is* universally MFA *if* P *is MFA w.r.t.* I^*_P.

Before we embark on analysing the computational properties of MFA we demonstrate the following proposition, which will be used later on.

Proposition 19. *For each set of rules* P, *we have that* P *is universally MFA if and only if* P *is MFA w.r.t. every set of facts* F.

Proof. Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$; we first show the following lemma.

Lemma 20. *Let* P *be a set of rules and let* F *be a set of facts. For each* $n \geq 0$, *there exists an extended homomorphism from* F^n_Q *to* $(I^*_P)^n_Q$.

Proof. We prove the claim by induction on n.

For $n = 0$, we define an extended homomorphism σ as follows: for each term $\mathsf{t} \in \mathsf{Terms}(F^0_Q)$, we set $\sigma(\mathsf{t}) = \mathsf{t}$ if $\mathsf{t} \in \mathsf{Const}(P)$ and $\sigma(\mathsf{t}) = *$ otherwise; for such σ, we have $F^0_Q\sigma \subseteq (I^*_P)^0_Q$.

Next, we assume that the claim holds for $n = k$ and we prove it for $n = k+1$. By induction hypothesis, there exists an extended homomorphism σ from F^k_Q to $(I^*_P)^k_Q$. Let σ' be the extended substitution from $\mathsf{Terms}(F^{k+1}_Q)$ to $\mathsf{Terms}((I^*_P)^{k+1}_Q)$ defined as follows:

- If $\mathsf{t} \in \mathsf{Terms}(F_Q^k)$, then $\sigma'(\mathsf{t}) = \sigma(\mathsf{t})$.
- If $\mathsf{t} \in \mathsf{Terms}(F_Q^{k+1} \setminus F_Q^k)$, then t is of the form $\mathsf{f}(\vec{\mathsf{t}})$ and $\sigma'(\mathsf{f}(\vec{\mathsf{t}})) = \mathsf{f}(\vec{\mathsf{t}}\sigma)$.

We show that σ' is an extended homomorphism from F_Q^{k+1} to $(I_P^*)_Q^{k+1}$.

- If $\alpha \in F_Q^k$, then $\alpha\sigma \in (I_P^*)_Q^k \subseteq (I_P^*)_Q^{k+1}$.
- If $\alpha \in F_Q^{k+1} \setminus F_Q^k$, then there exists a rule $r \in Q$ of the form $\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}}) \rightarrow \mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{f}}(\vec{\mathsf{x}}))$ and substitution θ such that $\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}})\theta \subseteq F_Q^k$, $\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{f}}(\vec{\mathsf{x}}))\theta \not\subseteq F_Q^k$ and $\alpha \in \mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{f}}(\vec{\mathsf{x}}))\theta$. By induction hypothesis, $(\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}})\theta)\sigma \subseteq (I_P^*)_Q^k$, which by definition of chase implies $(\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{f}}(\vec{\mathsf{x}}))\theta)\sigma \subseteq (I_P^*)_Q^{k+1}$ and yields $\alpha\sigma' \in (I_P^*)_Q^{k+1}$ as desired.

□

We are now ready to prove Proposition 19.

Assume for a contradiction that P is universally MFA and there exists a set of facts F such that P is not MFA w.r.t. F. Then $\mathsf{Chase}_Q(F)$ contains Cycle which by Lemma 20 implies that $\mathsf{Cycle} \in \mathsf{Chase}_Q(I_P^*)$ and gives us the required contradiction.

Assume that P is not universally MFA. Then, P is not MFA w.r.t. the critical instance I_P^*, which implies that there exists at least one set of facts F such that P is not MFA w.r.t. F, where F is the critical instance. □

Example 21. Let us revisit Example 5, which lists a set of rules for which chase terminates but which is not found acyclic by any of the conditions introduced in Section 3.3.1. We have that $\mathsf{MFA}(r_i) = r_i$ for $i = 1, 3, 5, 6, 7$ because these are datalog rules; moreover, we set $r_2' = \mathsf{MFA}(r_2)$ and $r_4' = \mathsf{MFA}(r_4)$.

$$\mathsf{botox}(\mathsf{x}_1) \rightarrow \mathsf{highlyToxic}(\mathsf{x}_1) \wedge \mathsf{metabolite}(\mathsf{x}_1) \qquad (r_1)$$

$$\mathsf{highlyToxic}(\mathsf{x}_2) \rightarrow \exists \mathsf{y}_2.\mathsf{hasProperty}(\mathsf{x}_2, \mathsf{y}_2) \wedge \mathsf{highToxicity}(\mathsf{y}_2) \wedge \mathsf{N}^2_{r_2}(\mathsf{y}_2) \wedge \mathsf{Succ}(\mathsf{x}_2, \mathsf{y}_2) \qquad (r_2')$$

$$\mathsf{hasProperty}(\mathsf{x}_3, \mathsf{z}_3) \wedge \mathsf{highToxicity}(\mathsf{z}_3) \rightarrow \mathsf{highlyToxic}(\mathsf{x}_3) \qquad (r_3)$$

$$\mathsf{highToxicity}(\mathsf{x}_4) \rightarrow \exists \mathsf{y}_4.\mathsf{hasToxicityDegree}(\mathsf{x}_4, \mathsf{y}_4) \wedge \mathsf{toxicityDegreeOne}(\mathsf{y}_4) \wedge \mathsf{N}^4_{r_4}(\mathsf{y}_4) \wedge \mathsf{Succ}(\mathsf{x}_4, \mathsf{y}_4) \qquad (r'_4)$$

$$\mathsf{hasToxicityDegree}(\mathsf{x}_5, \mathsf{w}_5) \rightarrow \mathsf{hasProperty}(\mathsf{x}_5, \mathsf{w}_5) \qquad (r_5)$$

$$\mathsf{metabolite}(\mathsf{x}_6) \wedge \mathsf{hasProperty}(\mathsf{x}_6, \mathsf{z}_6) \wedge \mathsf{hasToxicityDegree}(\mathsf{z}_6, \mathsf{w}_6) \rightarrow \mathsf{toxin}(\mathsf{x}_6) \qquad (r_6)$$

$$\mathsf{toxin}(\mathsf{x}_7) \wedge \mathsf{hasProperty}(\mathsf{x}_7, \mathsf{z}_7) \wedge \mathsf{hasProperty}(\mathsf{z}_7, \mathsf{w}_7) \wedge \mathsf{toxicityDegreeOne}(\mathsf{w}_7) \rightarrow \mathsf{lethalToxin}(\mathsf{x}_7) \qquad (r_7)$$

Let $P_\mathsf{b} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ and $F = \{\mathsf{botox(b)}\}$; $\mathsf{MFA}(P_\mathsf{b})$ consists of the rules in $\{\mathsf{MFA}(r_i)\}_{i=1}^7$, rule (4.1), rule (4.2) and rule (4.3) instantiated for $\mathsf{N}^2_{r_2}$ and $\mathsf{N}^4_{r_4}$. In order to check whether P_b is MFA w.r.t. F it is sufficient to compute $\mathsf{Chase}_{Q_\mathsf{b}}(F)$, where $Q_\mathsf{b} = \mathsf{sk}(\mathsf{MFA}(P_\mathsf{b}))$, up to a step where the computation stops or Cycle is derived. Let f and g be the function symbols used to skolemise y_2 and y_4.

$$F^0_{Q_\mathsf{b}} = F = \{\mathsf{botox(b)}\}$$

$$F^1_{Q_\mathsf{b}} = F^0_{Q_\mathsf{b}} \cup \{\mathsf{highlyToxic(b)}, \mathsf{metabolite(b)}\}$$

$$F^2_{Q_\mathsf{b}} = F^1_{Q_\mathsf{b}} \cup \{\mathsf{hasProperty(b, f(b))}, \mathsf{highToxicity(f(b))}, \mathsf{Succ(b, f(b))}, \mathsf{N}^2_{r_2}(\mathsf{f(b)})\}$$

$$F^3_{Q_\mathsf{b}} = F^2_{Q_\mathsf{b}} \cup \{\mathsf{hasToxicityDegree(f(b), g(f(b)))}, \mathsf{toxicityDegreeOne(g(f(b)))}, \mathsf{N}^4_{r_4}(\mathsf{g(f(b))})\} \cup \{\mathsf{Succ(f(b), g(f(b)))}, \mathsf{Desc(b, f(b))}\}$$

$$F^4_{Q_\mathsf{b}} = F^3_{Q_\mathsf{b}} \cup \{\mathsf{hasProperty(f(b), g(f(b)))}, \mathsf{toxin(b)}, \mathsf{Desc(f(b), g(f(b)))}, \mathsf{Desc(b, g(f(b)))}\}$$

$$F^5_{Q_\mathsf{b}} = F^4_{Q_\mathsf{b}} \cup \{\mathsf{lethalToxin(b)}\} = \mathsf{Chase}_{Q_\mathsf{b}}(F)$$

Since $\mathsf{Cycle} \notin \mathsf{Chase}_{Q_\mathsf{b}}(F)$, P_b is MFA w.r.t. F. Moreover, P_b is universally MFA as it is indicated by the chase sequence of Q_b w.r.t. $I^*_{P_\mathsf{b}}$:

$$(I^*_{P_\mathsf{b}})^0_{Q_\mathsf{b}} = I^*_{P_\mathsf{b}} = \{\mathsf{botox}(*), \mathsf{highlyToxic}(*), \mathsf{metabolite}(*), \mathsf{hasProperty}(*,*)\} \cup \{\mathsf{highToxicity}(*), \mathsf{hasToxicityDegree}(*,*), \mathsf{toxicityDegreeOne}(*)\} \cup \{\mathsf{toxin}(*), \mathsf{lethalToxin}(*,*)\}$$

$$
\begin{aligned}
(I^*_{P_\mathsf{b}})^1_{Q_\mathsf{b}} = \; & (I^*_{P_\mathsf{b}})^0_{Q_\mathsf{b}} \cup \{\mathsf{hasProperty}(*, \mathsf{f}(*)), \mathsf{Succ}(*, \mathsf{f}(*)), \mathsf{highToxicity}(\mathsf{f}(*)), \mathsf{N}^2_{r_2}(\mathsf{f}(*))\} \\
& \cup \{\mathsf{hasToxicityDegree}(*, \mathsf{g}(*)), \mathsf{Succ}(*, \mathsf{g}(*)), \mathsf{toxicityDegreeOne}(\mathsf{g}(*))\} \\
& \cup \{\mathsf{N}^4_{r_4}(\mathsf{g}(*))\} \\
(I^*_{P_\mathsf{b}})^2_{Q_\mathsf{b}} = \; & (I^*_{P_\mathsf{b}})^1_{Q_\mathsf{b}} \cup \{\mathsf{hasProperty}(*, \mathsf{g}(*)), \mathsf{hasToxicityDegree}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*)))\} \\
& \cup \{\mathsf{toxicityDegreeOne}(\mathsf{g}(\mathsf{f}(*)))\} \\
& \cup \{\mathsf{Succ}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*))), \mathsf{N}^4_{r_4}(\mathsf{g}(\mathsf{f}(*))), \mathsf{Desc}(*, \mathsf{f}(*)), \mathsf{Desc}(*, \mathsf{g}(*))\} \\
(I^*_{P_\mathsf{b}})^3_{Q_\mathsf{b}} = \; & (I^*_{P_\mathsf{b}})^2_{Q_\mathsf{b}} \cup \{\mathsf{hasProperty}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*))), \mathsf{Desc}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*)))\} \\
(I^*_{P_\mathsf{b}})^4_{Q_\mathsf{b}} = \; & (I^*_{P_\mathsf{b}})^3_{Q_\mathsf{b}} \cup \{\mathsf{Desc}(*, \mathsf{g}(\mathsf{f}(*)))\} = \mathsf{Chase}_{Q_\mathsf{b}}(I^*_{P_\mathsf{b}})
\end{aligned}
$$

♢

A preliminary version of MFA appeared in our previous work under the name *semantic acyclicity*; however, that approach suffered from the disadvantage of not being able to handle positive existential rules of an arbitrary shape and of requiring the existence of an ad-hoc ordering among the rules in order to determine acyclicity [175]. These deficiencies were overcome by introducing an improved notion of acyclicity which is MFA. Moreover, MFA has the benefit of allowing to merge the acyclicity check with the computation of the model, which implies that if a set of rules P is acyclic w.r.t. a set of facts F then reasoning comes 'for free': the model of $P \cup F$ can be easily extracted from $\mathsf{Chase}_{\mathsf{sk}(\mathsf{MFA}(P))}(F)$. In Section 4.3 we compare MFA with existing acyclicity conditions and show that MFA strictly captures many of them. The following proposition shows that MFA characterises the runs of the skolem chase that do not give rise to a cyclic term; this property will serve as the basis for establishing complexity bounds for checking MFA.

Proposition 22. *A set of rules P is MFA w.r.t. a set of facts F if and only if the chase of $\mathsf{sk}(\mathsf{MFA}(P))$ w.r.t. F does not contain a cyclic term.*

Proof. Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$ and let $F^0_Q, F^1_Q, \ldots$ be the chase sequence of Q w.r.t. F; furthermore, let $\mathsf{f}^{\mathsf{i}}_r$ be the function symbol used to skolemise the i-th existentially quantified variable in rule r. We next prove that the following claims hold for each $k \geq 0$ and for $k = \infty$, where $F^\infty_Q = \bigcup_{k \geq 0} F^k_Q = \mathsf{Chase}_Q(F)$.

1. For each term t of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ occurring in F^k_Q, we have $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}) \in F^k_Q$; vice versa, if $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}) \in \mathsf{Chase}_Q(F)$, then t is of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$.

2. For each term t of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ occurring in F^k_Q and each term $\mathsf{t}' \in \vec{\mathsf{t}}$, we have $\mathsf{Succ}(\mathsf{t}', \mathsf{t}) \in F^k_Q$; vice versa, if $\mathsf{Succ}(\mathsf{t}', \mathsf{t}) \in \mathsf{Chase}_Q(F)$, then t is of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ and $\mathsf{t}' \in \vec{\mathsf{t}}$.

3. For all terms t and t' occurring in F^k_Q such that t' is a proper subterm of t, we have $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in F^{k+1}_Q$; conversely, $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in \mathsf{Chase}_Q(F)$ implies that t' is a proper subterm of t.

(Claims 1 and 2, first part for $k \geq 0$) We prove them by induction on k. Since F^0_Q does not contain functional terms, the claims hold trivially for $k = 0$. For the induction step, we assume that both claims hold for F^k_Q and we show the claims for F^{k+1}_Q. Since $F^k_Q \subseteq F^{k+1}_Q$, both claims clearly hold for each term t that occurs in F^k_Q. Consider an arbitrary term t of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ that does not occur in F^k_Q, and that occurs in F^{k+1}_Q and an arbitrary term $\mathsf{t}' \in \vec{\mathsf{t}}$. There exists $r \in P$ such that t is introduced into F^{k+1}_Q by an application of $\mathsf{sk}(\mathsf{MFA}(r))$. Since the head of $\mathsf{sk}(\mathsf{MFA}(r))$ contains atoms $\mathsf{N}^{\mathsf{i}}_r(\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{x}}))$ and $\mathsf{Succ}(\mathsf{x}_\mathsf{j}, \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{x}}))$ for each $\mathsf{x}_\mathsf{j} \in \vec{\mathsf{x}}$, we have $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}) \in F^{k+1}_Q$ and $\mathsf{Succ}(\mathsf{t}', \mathsf{t}) \in F^{k+1}_Q$ for each $\mathsf{t}' \in \vec{\mathsf{t}}$.

(Claims 1 and 2, second part) Each predicate $\mathsf{N}^{\mathsf{i}}_r$ and the predicate Succ occur in Q only in the head of some rule $\mathsf{sk}(\mathsf{MFA}(r))$ and in the form $\mathsf{N}^{\mathsf{i}}_r(\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{x}}))$ and $\mathsf{Succ}(\mathsf{x},\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{x}}))$, where $\mathsf{x} \in \vec{\mathsf{x}}$, which establishes the claim.

(Claim 3, first part for $k \geq 0$) We prove it by induction on k. Since F^0_Q does not contain functional terms, the claim follows directly for $k = 0$. Assume now that the claim holds for some k and consider an arbitrary term $\mathsf{t} = \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ occurring in F^k_Q such that t' is a subterm of some $\mathsf{t}_\mathsf{j} \in \vec{\mathsf{t}}$. By Claim 2, we have $\mathsf{Succ}(\mathsf{t}_\mathsf{j}, \mathsf{t}) \in F^k_Q$. We distinguish two cases: if $\mathsf{t}' \in \vec{\mathsf{t}}$, then $\mathsf{t}_\mathsf{j} = \mathsf{t}'$ and $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in F^{k+1}_Q$ by rule (4.1). Otherwise, t' is a proper subterm of some $\mathsf{t}_\mathsf{j} \in \vec{\mathsf{t}}$ and since t' and t_j both occur in F^{k-1}_Q by induction hypothesis $\mathsf{Desc}(\mathsf{t}', \mathsf{t}_\mathsf{j}) \in F^k_Q$; the latter implies $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in F^{k+1}_Q$ due to rule (4.2).

(Claim 3, second part) The 'proper subterm' relation is transitive and rules (4.1)

and (4.2) effectively define Desc as the transitive closure of Succ, which implies the claim, using the second part of Claim 2.

(Claims 1, 2 and 3, first part for $k = \infty$) Let t be a term of the form $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$ occurring in $\mathsf{Chase}_Q(F)$ and let t' be a term such that $\mathsf{t}' \in \vec{\mathsf{t}}$. Since t occurs in $\bigcup_{k\geq 0} F^k_Q$, there exists $k \geq 0$ such that t and t' occur in F^k_Q and, so, by Claims 1 and 2 and since $F^k_Q \subseteq \mathsf{Chase}_Q(F)$ for each $k \geq 0$, we have $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}) \in \mathsf{Chase}_Q(F)$ and $\mathsf{Succ}(\mathsf{t}', \mathsf{t}) \in \mathsf{Chase}_Q(F)$. Similarly, for all terms t' and t, if t' is a proper subterm of t and they occur in $\mathsf{Chase}_Q(F)$, then there exists $k \geq 0$ such that t and t' occur in F^k_Q, which by Claim 3, implies $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in F^{k+1}_Q$ and, so, $\mathsf{Desc}(\mathsf{t}', \mathsf{t}) \in \mathsf{Chase}_Q(F)$.

We are now ready to prove the main claim. If $\mathsf{Chase}_Q(F)$ contains a cyclic term, then there exist terms t_1 and t_2 such that $\mathsf{t}_1 = \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{s}})$ is a subterm of t and $\mathsf{t}_2 = \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{u}})$ is a proper subterm of t_1. By Claims 1 and 3, these statements imply $\{\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_2), \mathsf{Desc}(\mathsf{t}_2, \mathsf{t}_1), \mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_1)\} \subseteq \mathsf{Chase}_Q(F)$. Since Q contains rule (4.3), the latter inclusion implies $\mathsf{Cycle} \in \mathsf{Chase}_Q(F)$, which holds if and only P is not MFA w.r.t. F. For the converse claim, assume that P is not MFA w.r.t. a set of facts F. Then, by Definition 18 we have that $F \cup \mathsf{MFA}(P) \models \mathsf{Cycle}$. Since the special nullary predicate Cycle occurs only on the right-hand side of rule (4.3), there exist terms t_1 and t_2, a rule $r \in P$, and a predicate $\mathsf{N}^{\mathsf{i}}_r$ such that $\{\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_1), \mathsf{Desc}(\mathsf{t}_1, \mathsf{t}_2), \mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_2)\} \subseteq \mathsf{Chase}_Q(F)$. Since $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_1)$ and $\mathsf{N}^{\mathsf{i}}_r(\mathsf{t}_2)$ are contained in $\mathsf{Chase}_Q(F)$, Claim 1 implies that t_1 and t_2 are of the form $\mathsf{t}_1 = \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{u_1}})$ and $\mathsf{t}_2 = \mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{u_2}})$, respectively. Finally, $\mathsf{Desc}(\mathsf{t}_1, \mathsf{t}_2) \in \mathsf{Chase}_Q(F)$ and Claim 3 imply that t_1 is a proper subterm of t_2, so $\mathsf{Chase}_Q(F)$ contains a cyclic term. □

We next show that this characterisation imposes a doubly exponential bound on the number of distinct terms and atoms that can be generated during execution of the skolem chase over an MFA set of rules. Consequently, this bound implies termination of the skolem chase over MFA rules in 2ExpTime. This bound is not a surprising one, given that boolean conjunctive query answering over a weakly-acyclic set of rules has been proved to be 2ExpTime-complete [44].

Proposition 23. *For a set of rules P and a set of facts F, if P is MFA w.r.t. F,*

then $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ *can be computed in doubly exponential time in the size of* $P \cup F$.

Proof. Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$, let a be the maximum arity of a function symbol in Q (we assume that $a > 1$, the special case $a = 1$ is studied later), let b, c and d be the number of distinct predicate, constant and function symbols, respectively, occurring in Q and let e be the maximum arity of a predicate symbol in Q. Consider now an arbitrary term t occurring in $\mathsf{Chase}_Q(F)$; the term t can be seen as a tree with branching factor a labelled by constants in the leaf nodes and function symbols in the internal nodes; furthermore, since t is not cyclic by Proposition 22, $\mathsf{depth}(\mathsf{t}) \leq d$, so the tree has at most a^d leaves and a^d inner nodes. Thus, the number of different terms occurring in $\mathsf{Chase}_Q(F)$ is bounded by $\ell = d^{a^d} \cdot c^{a^d} = (d \cdot c)^{a^d}$ and the number of different atoms in $\mathsf{Chase}_Q(F)$ is bounded by $b \cdot \ell^e = b \cdot ((c \cdot d)^{a^d})^e = b \cdot (c \cdot d)^{e \cdot a^d}$, which makes the maximum size of $\mathsf{Chase}_Q(F)$ doubly exponential in the size of P and F. Moreover, for an arbitrary set of facts F' and skolemised rule r, the set $r(F')$ can be computed by examining all mappings of the variables in r to the terms occurring in F', which can be done in exponential time in the size of r and polynomial time in the size of F'. Therefore, $\mathsf{Chase}_Q(F)$ can be computed in time that is doubly exponential in the size of P and F. Finally, one can verify that $\mathsf{Chase}_{\mathsf{sk}(P)}(F) \subseteq \mathsf{Chase}_Q(F)$, so $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ can be computed in doubly exponential time as well. □

We subsequently use Proposition 23 to prove that checking MFA for a program P and a set of facts F is in 2ExpTime; moreover, we show that checking whether a program P is universally MFA is 2ExpTime-hard. The two results combined imply that for a program P deciding whether P is universally MFA and whether P is MFA w.r.t. a set of facts F is 2ExpTime-complete. This is because, if checking whether a program P is MFA w.r.t. a set of facts F is in 2ExpTime, then checking whether P is universally MFA is also in 2ExpTime. Similarly, if we can solve an 2ExpTime-hard problem by reducing it to checking whether a program P is universally MFA, we can also solve it by checking whether P is MFA w.r.t. a set of facts F (i.e. the critical instance).

Theorem 24. *For a set of rules* P *and a set of facts* F*, checking whether* P *is*

MFA w.r.t. F is in 2ExpTime *for combined complexity and checking whether P is universally MFA is* 2ExpTime*-hard for program complexity.*

Proof. (Membership) Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$, let $F_Q^0, F_Q^1, \ldots$ be the chase sequence for Q w.r.t. F, and let ℓ, b and e be as defined in the proof of Proposition 23. As we already saw, the number of different atoms that can be constructed from ℓ terms is bounded by $k = b \cdot \ell^e$. Let $k' = k + 3$; we next show that one can decide whether P is MFA w.r.t. F by constructing $F_Q^{k'}$ and then checking whether $\mathsf{Cycle} \in F_Q^{k'}$. As shown in the proof of Proposition 23, $F_Q^{k'}$ can be computed in doubly exponential time in the size of P and F.

If $F_Q^k = F_Q^{k'}$, then $\mathsf{Chase}_F(Q) = F_Q^k$, so P is MFA if and only if $\mathsf{Cycle} \in F_Q^k$. Otherwise, we have $F_Q^k \subsetneq F_Q^{k'}$; but then, F_Q^{k+1} contains at least one cyclic term t of the form $\mathsf{f}_r^{\mathsf{i}}(\vec{\mathsf{t}})$ such that there exists $\mathsf{t}_\mathsf{j} \in \vec{\mathsf{t}}$ that has a subterm t' of the form $\mathsf{f}_r^{\mathsf{i}}(\vec{\mathsf{s}})$. Since F_Q^{k+1} satisfies Claims 1–3 from the proof of Proposition 22, we have $\{\mathsf{N}_r^{\mathsf{i}}(\mathsf{t}'), \mathsf{Desc}(\mathsf{t}', \mathsf{t}), \mathsf{N}_r^{\mathsf{i}}(\mathsf{t})\} \subseteq F_Q^{k+2}$; by rule (4.3) and the fact that rules without functional terms are applied before rules with functional terms, we have $\mathsf{Cycle} \in F_Q^{k'}$; thus, $\mathsf{Cycle} \in \mathsf{Chase}_Q(F)$, so P is not MFA.

(Hardness) In order to establish hardness, we draw upon a result by Calì et al. according to which query answering over a weakly acyclic set of rules is 2ExpTime-hard [44]. Specifically, Calì et al. prove that for a weakly acyclic set of rules P, a set of facts F and a Boolean conjunctive query q, deciding $F \cup P \models q$ is 2ExpTime-hard. We demonstrate our claim by reducing the problem of checking $F \cup P \models q$ for such P, F and q to checking whether a program P' is universally MFA. We next construct such P'.

For a constant c, let v_c be a fresh variable unique for c, let $\vec{\mathsf{v}}_\mathsf{c}$ be the vector of all variables corresponding to constants in F, let w' be a variable and let $\mathsf{F}(\vec{\mathsf{v}}_\mathsf{c})$ be defined as follows:

$$\mathsf{F}(\vec{\mathsf{v}}_\mathsf{c}) = \bigwedge_{\mathsf{P}(\mathsf{c}_1,\ldots,\mathsf{c}_\mathsf{k}) \in F} \hat{\mathsf{P}}(\mathsf{w}', \mathsf{v}_{\mathsf{c}_1}, \ldots, \mathsf{v}_{\mathsf{c}_\mathsf{k}}) \tag{4.4}$$

Let q be of the form $\exists \mathsf{y}.\phi(\vec{\mathsf{y}})$, let U be a fresh unary predicate and let B be a fresh binary predicate. Rules r_1 and r_2 are defined as follows ($\mathsf{Aster}(\ldots)$ is defined in the

beginning of the chapter).

$$\mathsf{Aster}(\mathsf{w}, \phi(\vec{\mathsf{y}})) \rightarrow \mathsf{U}(\mathsf{w}) \tag{r_1}$$

$$\mathsf{U}(\mathsf{w}) \rightarrow \exists \mathsf{w}', \vec{\mathsf{v}}_\mathsf{c}.\mathsf{B}(\mathsf{w}, \mathsf{w}') \wedge \mathsf{F}(\vec{\mathsf{v}}_\mathsf{c}) \tag{r_2}$$

Let f and g_c be the function symbols used to skolemise w' and v_c for each $\mathsf{v}_\mathsf{c} \in \vec{\mathsf{v}}_\mathsf{c}$, respectively. We set $P' = \mathsf{Aster}(P) \cup \{r_1, r_2\}$.

We next show that $F \cup P \models q$ if and only if P' is not universally MFA by proving the following two equivalences:

$$F \cup P \models q \quad \Leftrightarrow \mathsf{U}(\mathsf{f}(*)) \in \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'}) \tag{4.5}$$

$$\mathsf{U}(\mathsf{f}(*)) \in \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'}) \quad \Leftrightarrow P' \text{ is not universally MFA} \tag{4.6}$$

We first show (4.5). We have $U(\mathsf{f}(*)) \in \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$ if and only if there exists a substitution θ such that $\theta(\mathsf{w}) = \mathsf{f}(*)$ and $\mathsf{Aster}(\mathsf{w}, \phi(\vec{\mathsf{y}}))\theta \subseteq \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$. Due to the definition of $\mathsf{Aster}(\mathsf{w}, \phi(\vec{\mathsf{y}}))$ the latter is true if and only if the substitution θ is a homomorphism from $\mathsf{Aster}(\mathsf{f}(*), \phi(\vec{\mathsf{y}}))$ to $\mathsf{Chase}_{\mathsf{sk}(\mathsf{Aster}(P))}(F(\vec{\mathsf{v}}_\mathsf{c}\sigma))$, where $\sigma = \{\mathsf{w}' \mapsto \mathsf{f}(*)\} \cup \{\mathsf{v}_\mathsf{c} \mapsto \mathsf{g}_\mathsf{c}(*)\}_{\mathsf{v}_\mathsf{c} \in \vec{\mathsf{v}}_\mathsf{c}}$. The latter holds if and only if there exists a homomorphism from $\phi(\vec{\mathsf{y}})$ to $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$, which is equivalent to $F \cup P \models q$.

Finally, we demonstrate (4.6) in two steps.

Assume that $\mathsf{U}(\mathsf{f}(*)) \notin \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$. Then, the term $\mathsf{f}(\mathsf{f}(*))$ does not occur in $\mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$. Moreover, since P is weakly acyclic, P is super-weakly acyclic [177] and as implied by Theorems 34 and 55 (that we show later), we also have that P is universally MFA. Due to the definition of $\mathsf{Aster}(P)$ this implies that $\mathsf{Aster}(P)$ is universally MFA, that is $\mathsf{Chase}_{\mathsf{sk}(\mathsf{Aster}(P))}(I^*_{P'})$ does not contain a cyclic term either. Since $\{\mathsf{U}, \mathsf{B}\} \cap \mathsf{Pred}(\mathsf{Aster}(P)) = \emptyset$ and $\mathsf{U}(\mathsf{f}(*)) \notin \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$, rules r_1 and r_2 are not applied for a substitution σ such that $\sigma(\mathsf{w}) = \mathsf{f}(*)$ and, so, $\mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$ does not contain a cyclic term either. By Proposition 22, the latter implies that P' is universally MFA.

If $\mathsf{U}(\mathsf{f}(*)) \in \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$, then by r_2 we have $B(\mathsf{f}(*), \mathsf{f}(\mathsf{f}(*))) \in \mathsf{Chase}_{\mathsf{sk}(P')}(I^*_{P'})$, which by Proposition 22 implies that P' is not universally MFA. □

Corollary 25. *Let P be a set of rules such that the predicates in P are of bounded*

arity and let F be a set of facts. Checking whether P is MFA w.r.t. F is in 2ExpTime *for combined complexity and checking whether P is universally MFA is* 2ExpTime*-hard for program complexity.*

Proof. Membership follows by repeating the membership argument for Theorem 24 and observing that for e being constant, the bound on the number of atoms remains doubly exponential. Hardness is a direct consequence of the fact that Calì et al. [44] prove that query answering under a weakly-acyclic set of rules is 2ExpTime-hard even in the case of bounded arity. □

The complexity results of Theorem 24 are not particularly encouraging since the majority of existing acyclicity conditions can be checked in P or NP. In order to mitigate this problem, we suggest two solutions. First, we single out a fragment of existential rules for which the cost of checking MFA is lower. The second approach is to suitably modify MFA, which is presented in Section 4.1.2.

Definition 26 (∃-1 rule)**.** *A rule r is an ∃-1 rule if $|\mathsf{fr}(r)| \leq 1$ or if r is a datalog rule. A program P is an ∃-1 program if each rule in P is ∃-1.*

Since ∃-1 rules have frontier of size at most one, the arity of function symbols after skolemisation is at most one, which—as we show next—reduces the computational complexity by one exponential factor. Moreover, ∃-1 rules are a practically relevant class of existential rules since all of the examples presented so far involve ∃-1 programs.

Theorem 27. *For a set of ∃-1 rules P and a set of facts F, determining whether P is MFA w.r.t. F is in* ExpTime *w.r.t. combined complexity and determining whether P is universally MFA is* ExpTime*-hard, even if the arity of the predicates is bounded.*

Proof. (Membership) Let a, b, c, d and e be defined as in the proof of Proposition 23. For ∃-1 rules we have $a = 1$ which implies that the bound on the number of different terms becomes $c \cdot d^d$ and the bound on the number of different atoms becomes $k = b \cdot c^e \cdot d^{ed}$, which is exponential in the size of P and F, even if e is fixed. Similarly to the membership proof for Theorem 24, MFA can be checked by computing the chase sequence of $\mathsf{sk}(\mathsf{MFA}(P))$ w.r.t. F for k' steps, where $k' = k + 3$.

1. Initialisation: if $w = \alpha_0 \dots \alpha_m$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{next_k}(\mathsf{x_i}, \mathsf{x_{i+1}}) \rightarrow \mathsf{state}_{\mathsf{q}_0}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{symbol}_{\alpha_\mathsf{i}}(\mathsf{x}_0, \mathsf{x_i}) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}_{\mathsf{m}+1})$$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{y})$$

2. Transition rules: for all $\delta = \langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle \in \Delta$

$$\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{y}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{v}, \mathsf{v}') \rightarrow \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \quad \text{if } \mathsf{m} = l$$

$$\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{next_k}(\mathsf{v}, \mathsf{v}') \rightarrow \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \quad \text{if } \mathsf{m} = r$$

3. Inertia: for all $\alpha \in \Sigma$

$$\mathsf{next_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{nextt}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

$$\mathsf{next_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{nextt}(\mathsf{y}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

4. Acceptance: for each accepting state $\mathsf{q_a} \in Q$

$$\mathsf{state}_{\mathsf{q_a}}(\mathsf{v_k}) \wedge \bigwedge_{0 \le i < k} \mathsf{child_i}(\mathsf{v_i}, \mathsf{v_{i+1}}) \rightarrow \mathsf{accept}(\mathsf{v}_0)$$

$$\mathsf{accept}(\mathsf{v}) \rightarrow \exists \mathsf{u}_1, \mathsf{u}_2 . \mathsf{T}(\mathsf{v}, \mathsf{u}_1, \mathsf{u}_2) \wedge \mathsf{s}_0(\mathsf{u}_1) \wedge \mathsf{s}_0(\mathsf{u}_2) \wedge \mathsf{next}_0(\mathsf{u}_1, \mathsf{u}_2) \wedge \mathsf{min}_0(\mathsf{u}_1)$$

Figure 4.1: Program $P^{\mathsf{comp}}_{\mathcal{T},w}$ simulating computation of a DTM $\mathcal{T}$ over w

(Hardness) To prove ExpTime-hardness w.r.t. combined complexity, we show that for every deterministic Turing machine (DTM) $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$, word $w \in \Sigma^*$ and number $k \ge 1$, we can construct an $\exists$-1 program $P_{\mathcal{T},w,k}$ such that

$$\mathcal{T} \text{ accepts } w \text{ in time } 2^k \;\; \Leftrightarrow \;\; P_{\mathcal{T},w,k} \text{ is not universally MFA} \tag{4.7}$$

Let $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$ be a DTM such that the transition function Δ consists of tuples of the form $\langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle$, where $\mathsf{q}, \mathsf{q}' \in Q$, $\alpha, \alpha' \in \Sigma$ and $\mathsf{m} \in \{l, r\}$ depending on whether the head moves left or right.

First, given $k \ge 1$, we construct an $\exists$-1 program P^{exp}_k of polynomial size, which

encodes a chain of 2^k elements. We use this exponentially long chain to model both the tape of cells and the timeline of instants. P_k^{exp} contains the following rules for each $i \in \{0, \ldots, k-1\}$:

$$\mathsf{s_i}(\mathsf{x}) \rightarrow \exists \mathsf{y}_1, \mathsf{y}_2 . \ell_\mathsf{i}(\mathsf{x}, \mathsf{y}_1) \wedge \mathsf{r_i}(\mathsf{x}, \mathsf{y}_2) \qquad (r_i)$$

$$\ell_\mathsf{i}(\mathsf{x}_1, \mathsf{x}_2) \rightarrow \mathsf{s_{i+1}}(\mathsf{x}_2) \wedge \mathsf{child_i}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{r_i}(\mathsf{x}_1, \mathsf{x}_2) \rightarrow \mathsf{s_{i+1}}(\mathsf{x}_2) \wedge \mathsf{child_i}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\ell_\mathsf{i}(\mathsf{z}, \mathsf{x}_1) \wedge \mathsf{r_i}(\mathsf{z}, \mathsf{x}_2) \rightarrow \mathsf{next_{i+1}}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{r_i}(\mathsf{z}_1, \mathsf{x}_1) \wedge \ell_\mathsf{i}(\mathsf{z}_2, \mathsf{x}_2) \wedge \mathsf{next_i}(\mathsf{z}_1, \mathsf{z}_2) \rightarrow \mathsf{next_{i+1}}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{min_i}(\mathsf{z}) \wedge \ell_\mathsf{i}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{min_{i+1}}(\mathsf{x})$$

We extend P_k^{exp} with the following two rules, which transitively close the precedence relation that arises from the binary predicate $\mathsf{next_k}$ between the elements of the chain:

$$\mathsf{next_k}(\mathsf{x}_1, \mathsf{x}_2) \rightarrow \mathsf{nextt}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{nextt}(\mathsf{x}_1, \mathsf{z}) \wedge \mathsf{nextt}(\mathsf{z}, \mathsf{x}_2) \rightarrow \mathsf{nextt}(\mathsf{x}_1, \mathsf{x}_2)$$

Next, we build $P_{\mathcal{T},w}^{\mathsf{comp}}$ which simulates the computation of $\mathcal{T}$ on w. We use the following predicates for our encoding:

- $\mathsf{state_q}(\mathsf{v})$ for $\mathsf{q} \in Q$, when $\mathcal{T}$ is at state q at time v;
- $\mathsf{head}(\mathsf{v}, \mathsf{x})$, when the head of $\mathcal{T}$ is on cell x at time v;
- $\mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x})$ for $\alpha \in \Sigma$, when cell x contains symbol α at time v;
- $\mathsf{accept}(\mathsf{v})$, when the computation of $\mathcal{T}$ on w has reached an accepting state at some instant $\mathsf{v_n}$, such that v is the subterm of $\mathsf{v_n}$ with $\mathsf{depth}(\mathsf{v_n}) - \mathsf{depth}(\mathsf{v}) = \mathsf{n}$.

The rules of $P_{\mathcal{T},w}^{\mathsf{comp}}$ are shown in Figure 4.1; also, let $P_{\mathcal{T},w,k} = P_k^{\mathsf{exp}} \cup P_{\mathcal{T},w}^{\mathsf{comp}}$. One can easily check that $P_{\mathcal{T},w,k}$ is an $\exists$-1 program. Finally, the size of $P_{\mathcal{T},w,k}$ is polynomial in the size of $\mathcal{T}$, w and k and the predicates in $P_{\mathcal{T},w,k}$ are of bounded arity which causes the result to hold even if the arity of predicates from P is bounded.

We now prove that (4.7) holds. Before that, we set $P = P_{\mathcal{T},w,k}$ to simplify the notation; let also g_1, g_2, $\mathsf{f}_1^{\mathsf{i}}$, $\mathsf{f}_2^{\mathsf{i}}$ be the function symbols used to skolemise u_1 and u_2 in the second acceptance rule and y_1 and y_2 in rule r_i.

Assume that $\mathcal{T}$ accepts w; we prove that P is not universally MFA by showing that $\mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*)$ contains a cyclic term. By definition of I_P^*, we have $\mathsf{accept}(*) \in I_P^*$ and, so,

$$\{\mathsf{s}_0(\mathsf{g}_1(*)), \mathsf{s}_0(\mathsf{g}_2(*)), \mathsf{next}_0(\mathsf{g}_1(*), \mathsf{g}_2(*)), \mathsf{min}_0(\mathsf{g}_1(*))\} \subseteq \mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*).$$

By definition of P_k^{exp} the facts above give rise to a chain containing 2^k elements. Therefore, by definition of $P_{\mathcal{T},w}^{\mathsf{comp}}$, if $\mathcal{T}$ accepts w in time 2^k, then we have that $\{\mathsf{accept}(\mathsf{g}_1(*)), \mathsf{accept}(\mathsf{g}_2(*))\} \cap \mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*) \neq \emptyset$. By the head of the second acceptance rule, this implies that a cyclic term of the form $\mathsf{g}_1(\mathsf{g}_1(*))$ or $\mathsf{g}_2(\mathsf{g}_2(*))$ is generated which entails that P is not universally MFA.

Assume that $\mathcal{T}$ does not accept w; we prove that P is universally MFA. By Proposition 22 it is sufficient to prove that $\mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*)$ does not contain a cyclic term. Given that the only rules that can give rise to a cyclic term are the second acceptance rule and rule r_i for $i \in \{0, \ldots, k-1\}$, it is sufficient to prove that (i) $\{\mathsf{accept}(\mathsf{g}_1(*)), \mathsf{accept}(\mathsf{g}_2(*))\} \cap \mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*) = \emptyset$ and (ii) $\mathsf{s}_{\mathsf{i}}(\mathsf{t}) \notin \mathsf{Chase}_{\mathsf{sk}(P)}(I_P^*)$ for a term t that contains $\mathsf{f}_1^{\mathsf{i}}$ or $\mathsf{f}_2^{\mathsf{i}}$. By definition of $P_{\mathcal{T},w}^{\mathsf{comp}}$ and P_k^{exp} and the fact that all rules that appear in P are connected (in the same sense that graph rules can be connected on p. 38) (i) does not hold only when an atom of the form $\mathsf{state}_{\mathsf{q_a}}(\mathsf{f}_{\mathsf{i}_{k-1}}^{k-1} \ldots (\mathsf{f}_{\mathsf{i}_0}^0(\mathsf{g}_{\mathsf{i}}(*))) \ldots)$, where $\mathsf{q_a}$ is an accepting state and $\mathsf{i}_0, \ldots, \mathsf{i}_{k-1}, \mathsf{i} \in \{1, 2\}$, is derived. The latter is true only when $\mathcal{T}$ accepts w which shows (i). Due to the conjunction of atoms $\bigwedge_{0 \leq i < k} \mathsf{child}_{\mathsf{i}}(\mathsf{v}_{\mathsf{i}+1}, \mathsf{v}_{\mathsf{i}})$ in the body of the first acceptance rule, the only derived atoms whose predicate is s_0 are of the form $\mathsf{s}_0(*)$, $\mathsf{s}_0(\mathsf{g}_1(*))$ or $\mathsf{s}_0(\mathsf{g}_2(*))$ which combined with the definition of P_k^{exp} and the fact that $\mathcal{T}$ does not accept w implies (ii). □

4.1.2 Model Summarising Acyclicity

The high cost of checking MFA for a set of rules P is due to the fact that the arity of function symbols in $\mathsf{sk}(P)$ is unbounded and that the depth of cyclic terms can

be linear in P. To obtain an acyclicity condition that is easier to check, we can coarsen the structure used for cycle analysis. To this end, we next introduce *model-summarising acyclicity* (MSA), which 'summarises' the model of P by reusing the same constant to satisfy an existential quantifier instead of introducing terms with complex internal structure. We thus trade some faithfulness for tractability in order to design a condition that is more efficient to check.

Definition 28. *Let* Succ, Desc *and* $\mathsf{N}^{\mathsf{i}}_r$ *be as specified in Definition 18; furthermore, for each rule* $r = \mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}}) \rightarrow \exists \vec{\mathsf{y}}.\mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{y}})$ *and each variable* $\mathsf{y_i} \in \vec{\mathsf{y}}$*, let* $\mathsf{c}^{\mathsf{i}}_r$ *be a fresh constant unique for* $\mathsf{y_i}$ *and* r*. Then,* $\mathsf{MSA}(r)$ *is the following rule, where* θ_{MSA} *is the substitution that maps each variable* $\mathsf{y_i} \in \vec{\mathsf{y}}$ *to* $\mathsf{c}^{\mathsf{i}}_r$*:*

$$\mathsf{B}(\vec{\mathsf{x}}, \vec{\mathsf{z}}) \rightarrow \mathsf{H}(\vec{\mathsf{x}}, \vec{\mathsf{y}})\theta_{\mathsf{MSA}} \wedge \bigwedge_{\mathsf{y_i} \in \vec{\mathsf{y}}} \left[\mathsf{N}^{\mathsf{i}}_r(\mathsf{y_i})\theta_{\mathsf{MSA}} \wedge \bigwedge_{\mathsf{x_j} \in \vec{\mathsf{x}}} \mathsf{Succ}(\mathsf{x_j}, \mathsf{y_i})\theta_{\mathsf{MSA}} \right]$$

For a set of rules P*,* $\mathsf{MSA}(P)$ *is the smallest set that contains* $\mathsf{MSA}(r)$ *for each rule* $r \in P$*, rule* (4.1)*, rule* (4.2) *and rule* (4.3) *instantiated for each predicate* $\mathsf{N}^{\mathsf{i}}_r$*. Set* P *is* model-summarising acyclic *(MSA) w.r.t. a set of facts* F *if* $F \cup \mathsf{MSA}(P) \not\models \mathsf{Cycle}$*; furthermore,* P *is* universally MSA *if* P *is MSA w.r.t.* I^*_P*.*

Similarly to MFA, if a program is universally MSA, then it is guaranteed that it is MSA w.r.t. every set of facts.

Proposition 29. *For a set of rules* P*, we have that* P *is universally MSA if and only if* P *is MSA w.r.t. every set of facts* F*.*

Proof. The statement can be proved similarly to Proposition 19; the only difference is that when we define the extended substitution σ' for $\mathsf{t} \in \mathsf{Terms}(F^{k+1}_{\mathsf{MSA}(P)} \setminus F^{k}_{\mathsf{MSA}(P)})$ and the term t is of the form $\mathsf{c}^{\mathsf{i}}_r$, we set $\sigma'(\mathsf{t}) = \mathsf{c}^{\mathsf{i}}_r$. □

Example 30. In the previous Section, we demonstrated that the set of rules arising from Example 5 (which is not identified as acyclic by most known acyclicity conditions) is universally MFA. Next, we show that it is also universally MSA. We have that $\mathsf{MSA}(P_{\mathsf{b}})$ contains $r_1, r''_2, r_3, r''_4, r_5, r_6, r_7$ (where r''_2 and $''_4$ are defined next),

rule (4.1), rule (4.2) and rule (4.3) instantiated for $\mathsf{N}^2_{r_2}$ and $\mathsf{N}^4_{r_4}$.

$$\begin{aligned}
\mathsf{botox}(\mathsf{x}_1) &\rightarrow \mathsf{highlyToxic}(\mathsf{x}_1) \wedge \mathsf{metabolite}(\mathsf{x}_1) && (r_1)\\
\mathsf{highlyToxic}(\mathsf{x}_2) &\rightarrow \mathsf{hasProperty}(\mathsf{x}_2, \mathsf{c}^2_{\mathsf{r}_2}) \wedge \\
&\quad\ \mathsf{highToxicity}(\mathsf{c}^2_{\mathsf{r}_2}) \wedge \\
&\quad\ \mathsf{Succ}(\mathsf{x}_2, \mathsf{c}^2_{\mathsf{r}_2}) \wedge \mathsf{N}^2_{\mathsf{r}_2}(\mathsf{c}^2_{\mathsf{r}_2}) && (r''_2)\\
\mathsf{hasProperty}(\mathsf{x}_3, \mathsf{z}_3) \wedge \mathsf{highToxicity}(\mathsf{z}_3) &\rightarrow \mathsf{highlyToxic}(\mathsf{x}_3) && (r_3)\\
\mathsf{highToxicity}(\mathsf{x}_4) &\rightarrow \mathsf{hasToxicityDegree}(\mathsf{x}_4, \mathsf{c}^4_{\mathsf{r}_4}) \wedge \\
&\quad\ \mathsf{toxicityDegreeOne}(\mathsf{c}^4_{\mathsf{r}_4}) \wedge \\
&\quad\ \mathsf{Succ}(\mathsf{x}_4, \mathsf{c}^4_{\mathsf{r}_4}) \wedge \mathsf{N}^4_{\mathsf{r}_4}(\mathsf{c}^4_{\mathsf{r}_4}) && (r''_4)\\
\mathsf{hasToxicityDegree}(\mathsf{x}_5, \mathsf{w}_5) &\rightarrow \mathsf{hasProperty}(\mathsf{x}_5, \mathsf{w}_5) && (r_5)\\
\mathsf{metabolite}(\mathsf{x}_6) \wedge \mathsf{hasProperty}(\mathsf{x}_6, \mathsf{z}_6) \wedge & \\
\mathsf{hasToxicityDegree}(\mathsf{z}_6, \mathsf{w}_6) &\rightarrow \mathsf{toxin}(\mathsf{x}_6) && (r_6)\\
\mathsf{toxin}(\mathsf{x}_7) \wedge \mathsf{hasProperty}(\mathsf{x}_7, \mathsf{z}_7) \wedge & \\
\mathsf{hasProperty}(\mathsf{z}_7, \mathsf{w}_7) \wedge \mathsf{toxicityDegreeOne}(\mathsf{w}_7) &\rightarrow \mathsf{lethalToxin}(\mathsf{x}_7) && (r_7)
\end{aligned}$$

Let $R_\mathsf{b} = \mathsf{MSA}(P_\mathsf{b})$. By the chase sequence of R_b w.r.t. $I^*_{P_\mathsf{b}}$ below, we conclude that $\mathsf{Cycle} \notin \mathsf{Chase}_{R_\mathsf{b}}(I^*_{R_\mathsf{b}})$ and thus P_b is universally MSA.

$$\begin{aligned}
(I^*_{P_\mathsf{b}})^0_{R_\mathsf{b}} = \quad I^*_{P_\mathsf{b}} \quad &= \{\mathsf{botox}(*), \mathsf{highlyToxic}(*), \mathsf{metabolite}(*), \mathsf{hasProperty}(*,*)\}\\
&\cup \{\mathsf{highToxicity}(*), \mathsf{hasToxicityDegree}(*,*), \mathsf{toxicityDegreeOne}(*)\}\\
&\cup \{\mathsf{toxin}(*), \mathsf{lethalToxin}(*,*)\}\\
(I^*_{P_\mathsf{b}})^1_{R_\mathsf{b}} = \quad (I^*_{P_\mathsf{b}})^0_{R_\mathsf{b}} &\cup \{\mathsf{hasProperty}(*, \mathsf{c}^2_{\mathsf{r}_2}), \mathsf{highToxicity}(\mathsf{c}^2_{\mathsf{r}_2}), \mathsf{Succ}(*, \mathsf{c}^2_{\mathsf{r}_2}), \mathsf{N}^2_{\mathsf{r}_2}(\mathsf{c}^2_{\mathsf{r}_2})\}\\
&\cup \{\mathsf{hasToxicityDegree}(*, \mathsf{c}^4_{\mathsf{r}_4}), \mathsf{toxicityDegreeOne}(\mathsf{c}^4_{\mathsf{r}_4}), \mathsf{Succ}(*, \mathsf{c}^4_{\mathsf{r}_4})\}\\
&\cup \{\mathsf{N}^4_{\mathsf{r}_4}(\mathsf{c}^4_{\mathsf{r}_4})\}\\
(I^*_{P_\mathsf{b}})^2_{R_\mathsf{b}} = \quad (I^*_{P_\mathsf{b}})^1_{R_\mathsf{b}} &\cup \{\mathsf{hasProperty}(*, \mathsf{c}^4_{\mathsf{r}_4}), \mathsf{hasToxicityDegree}(\mathsf{c}^2_{\mathsf{r}_2}, \mathsf{c}^4_{\mathsf{r}_4})\}\\
&\cup \{\mathsf{Succ}(\mathsf{c}^2_{\mathsf{r}_2}, \mathsf{c}^4_{\mathsf{r}_4}), \mathsf{Desc}(*, \mathsf{c}^2_{\mathsf{r}_2}), \mathsf{Desc}(*, \mathsf{c}^4_{\mathsf{r}_4})\}\\
(I^*_{P_\mathsf{b}})^3_{R_\mathsf{b}} = \quad (I^*_{P_\mathsf{b}})^2_{R_\mathsf{b}} &\cup \{\mathsf{hasProperty}(\mathsf{c}^2_{\mathsf{r}_2}, \mathsf{c}^4_{\mathsf{r}_4}), \mathsf{Desc}(\mathsf{c}^2_{\mathsf{r}_2}, \mathsf{c}^4_{\mathsf{r}_4}),\} = \mathsf{Chase}_{R_\mathsf{b}}(I^*_{P_\mathsf{b}})
\end{aligned}$$

$\diamondsuit$

Let us consider now what are the gains we obtain in terms of complexity. For a program P, the set $\mathsf{MSA}(P)$ consists exclusively of datalog rules. Thus, MSA can be checked using a datalog reasoner. This connection with datalog provides the upper complexity bound for checking MSA, which is ExpTime; the matching lower bound follows from the complexity of checking entailment of a ground atom by a datalog program as shown in the article by Dantsin et al. [65]. The complexity of datalog reasoning is $O(r \cdot n^v)$ where r is the number of rules, v is the maximum number of variables in a rule and n depends on the number of constants contained in the set of facts and rules; thus, checking MSA should be feasible if the rules in P contain a small number of variables. Theorem 31 pinpoints the complexity costs for checking MSA, both universally and w.r.t. a set of facts. The proof is similar to the proof of Theorem 24; the difference is that the complexity has dropped by one exponential factor due to the replacement of function symbols by constants in the MSA transformation.

Theorem 31. *For a set of rules P and a set of facts F, checking whether P is MSA w.r.t. F is in* ExpTime *for combined complexity and checking whether P is universally MSA is* ExpTime*-hard for program complexity.*

Proof. (Membership) Let $R = \mathsf{MSA}(P)$ and note that P is MSA w.r.t. F if and only if $F \cup P \not\models \mathsf{Cycle}$ which holds if and only if $\mathsf{Cycle} \notin \mathsf{Chase}_R(F)$. The total number of atoms occurring in $\mathsf{Chase}_R(F)$ is $b \cdot c^a$, where b is the number of predicates, c is the number of constants and a is the maximum arity of the predicates in R, which is exponential in the size of P and F. The rest of the membership proof is the same as in Theorem 24.

(Hardness) For a set of datalog rules P, a set of facts F and a ground atom α, checking whether $F \cup P \models \alpha$ is ExpTime-hard [65]. Let P' and F' be defined from P as in the hardness proof of Theorem 24; similarly to that proof one can show that P' is not universally MSA w.r.t. F' if and only if $F \cup P \models \alpha$. □

Moreover, a further drop in the cost of checking MSA can be achieved by fixing the arity of predicates, which is a reasonable assumption given that we have not

used a predicate with arity more than two in any of the so far motivating examples. Hardness for this case can be proved by repeating an argument similar to the one provided for Theorem 24.

Theorem 32. *For a set of rules P whose predicates are of bounded arity and a set of facts F, checking whether P is MSA w.r.t. F is in* coNP *for combined complexity and checking whether P is universally MSA is* coNP*-hard for program complexity.*

Proof. (Membership) Let $R = \mathsf{MSA}(P)$ and let a, b and c be defined as in the proof of Theorem 31. If a is bounded, then the number of ground atoms in $\mathsf{Chase}_R(F)$ becomes polynomial in the size of P and F. Furthermore, by the definition of the chase, $\mathsf{Cycle} \in \mathsf{Chase}_R(F)$ if and only if there exist a sequence of rules $r_1, \ldots, r_n$ of the form $r_i = \mathsf{B}_i \to \mathsf{H}_i$ and a sequence of substitutions $\sigma_1, \ldots, \sigma_n$ such that for each $1 \leq i \leq n$ we have $\mathsf{B}\sigma_i \subseteq F \cup \{\mathsf{H}_j\sigma_j \mid j < i\} \subseteq \mathsf{Chase}_R(F)$ and $\mathsf{H}_n\sigma_n = \mathsf{Cycle}$. By the bound on the size of $\mathsf{Chase}_R(F)$, we can assume that $n \leq b \cdot c^a$, where n is polynomial in the size of the program. Thus, we can guess the two sequences in polynomial time and check whether $\mathsf{B}\sigma_i \subseteq F \cup \{\mathsf{H}_j\sigma_j \mid j < i\} \subseteq \mathsf{Chase}_R(F)$ for each $1 \leq i \leq n$ in polynomial time. Therefore, $F \cup R \models \mathsf{Cycle}$ can be checked in nondeterministic polynomial time, so checking whether P is MSA w.r.t. F is in coNP.

(Hardness) Dantsin et al. [65] prove that checking $F \cup P \models \alpha$ for a set of datalog rules P with bounded arity, a set of facts F and a ground atom α is coNP-hard; NP-hardness thus follows from a reduction similar to the one used in the proof of Theorem 24. □

Next we see that, in contrast to the complexity for checking MFA, there is no improvement for checking acyclicity of ∃-1 programs; this is expected since $\mathsf{MSA}(P)$ no longer contains any function symbols and, so, restricting the arity of the function symbols does not bear any benefit. We obtain a tight complexity bound by following the same argument as the one in proof of Theorem 31.

Corollary 33. *For a set of ∃-1 rules P and a set of facts F, checking whether P is MSA w.r.t. F is in* ExpTime *for combined complexity and checking whether P is universally MSA is* ExpTime*-hard for program complexity.*

Proof. Both membership and hardness proofs are identical to the ones of Theorem 31. For the hardness case, the constructed program P' is $\exists$-1 because it contains only one existential rule with frontier of size 1. □

Before concluding this section, we present Theorem 34 and Example 35, which together show that MFA is strictly more general than MSA.

Theorem 34. *For a set of rules P and a set of facts F, if P is MSA w.r.t. F, then P is MFA w.r.t. F as well.*

Proof. Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$, let $R = \mathsf{MSA}(P)$ and let h be the extended substitution from $\mathsf{HU}(Q)$ to $\mathsf{HU}(R)$ such that $h(\mathsf{t}) = \mathsf{c}^{\mathsf{i}}_r$ if t is of the form $\mathsf{f}^{\mathsf{i}}_r(\ldots)$ and $h(\mathsf{t}) = \mathsf{t}$ if t is a constant. Note that R differs from Q only in that the former contains the constant $\mathsf{c}^{\mathsf{i}}_r$ in place of each functional term $\mathsf{f}^{\mathsf{i}}_r(\vec{\mathsf{t}})$. Thus, by an induction on i, one can show that $h(Q^i_F) \subseteq R^i_F$ for each i; this implies that h is an extended homomorphism from Q^i_F to R^i_F. Consequently, if $\mathsf{Cycle} \notin \mathsf{Chase}_R(F)$, then $\mathsf{Cycle} \notin \mathsf{Chase}_Q(F)$; hence, if P is MSA, then P is MFA as well, as required. □

Example 35. Let P consisting of the rules r_1, r_2, r_3 and r_4.

$$\mathsf{A}(\mathsf{x}) \rightarrow \exists \mathsf{y}_1.\mathsf{R}(\mathsf{x}, \mathsf{y}_1) \wedge \mathsf{B}(\mathsf{y}_1) \tag{r_1}$$

$$\mathsf{B}(\mathsf{x}) \rightarrow \exists \mathsf{y}_2.\mathsf{S}(\mathsf{x}, \mathsf{y}_2) \wedge \mathsf{T}(\mathsf{y}_2, \mathsf{x}) \tag{r_2}$$

$$\mathsf{A}(\mathsf{z}) \wedge \mathsf{S}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{C}(\mathsf{x}) \tag{r_3}$$

$$\mathsf{C}(\mathsf{z}) \wedge \mathsf{T}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{A}(\mathsf{x}) \tag{r_4}$$

Let $Q = \mathsf{sk}(\mathsf{MFA}(P))$ and let $R = \mathsf{MSA}(P)$; let also f and g be the function symbols used to skolemise y_1 and y_2 and let N_1, N_2, c_1 and c_2 be the corresponding predicates and constants needed for building $\mathsf{MFA}(P)$ and $\mathsf{MSA}(P)$. The chase sequence of Q

w.r.t. I^*_P appears below.

$$
\begin{aligned}
(I^*_P)^0_Q = \quad & I^*_P \qquad = \{\mathsf{A}(*), \mathsf{B}(*), \mathsf{C}(*), \mathsf{R}(*,*), \mathsf{S}(*,*), \mathsf{T}(*,*)\} \\
(I^*_P)^1_Q = \quad & (I^*_P)^0_Q \cup \{\mathsf{R}(*, \mathsf{f}(*)), \mathsf{B}(\mathsf{f}(*)), \mathsf{Succ}(*, \mathsf{f}(*)), \mathsf{N}_1(\mathsf{f}(*))\} \\
& \qquad \cup \{\mathsf{S}(*, \mathsf{g}(*)), \mathsf{T}(\mathsf{g}(*), *), \mathsf{Succ}(*, \mathsf{g}(*)), \mathsf{N}_2(\mathsf{g}(*))\} \\
(I^*_P)^2_Q = \quad & (I^*_P)^1_Q \cup \{\mathsf{S}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*))), \mathsf{T}(\mathsf{g}(\mathsf{f}(*)), \mathsf{f}(*)), \mathsf{Succ}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*))), \mathsf{N}_2(\mathsf{g}(\mathsf{f}(*)))\} \\
& \qquad \cup \{\mathsf{Desc}(*, \mathsf{f}(*)), \mathsf{Desc}(*, \mathsf{g}(*)), \mathsf{C}(\mathsf{g}(*))\} \\
(I^*_P)^3_Q = \quad & (I^*_P)^2_Q \cup \{\mathsf{Desc}(*, \mathsf{g}(\mathsf{f}(*))), \mathsf{Desc}(\mathsf{f}(*), \mathsf{g}(\mathsf{f}(*)))\} = \mathsf{Chase}_Q(I^*_P)
\end{aligned}
$$

By $\mathsf{Cycle} \notin \mathsf{Chase}_Q(I^*_P)$ we conclude that P is universally MFA; however, P is not universally MSA as shown next by the atom Cycle derived in the chase sequence of R w.r.t. I^*_P.

$$
\begin{aligned}
(I^*_P)^0_R = \quad & I^*_P \qquad = \{\mathsf{A}(*), \mathsf{B}(*), \mathsf{C}(*), \mathsf{R}(*,*), \mathsf{S}(*,*), \mathsf{T}(*,*)\} \\
(I^*_P)^1_R = \quad & (I^*_P)^0_R \cup \{\mathsf{R}(*, \mathsf{c}_1), \mathsf{B}(\mathsf{c}_1), \mathsf{Succ}(*, \mathsf{c}_1), \mathsf{N}_1(\mathsf{c}_1)\} \\
& \qquad \cup \{\mathsf{S}(*, \mathsf{c}_2), \mathsf{T}(\mathsf{c}_2, *), \mathsf{Succ}(*, \mathsf{c}_2), \mathsf{N}_2(\mathsf{c}_2)\} \\
(I^*_P)^2_R = \quad & (I^*_P)^1_R \cup \{\mathsf{Desc}(*, \mathsf{c}_1), \mathsf{Desc}(*, \mathsf{c}_2), \mathsf{S}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{T}(\mathsf{c}_2, \mathsf{c}_1), \mathsf{C}(\mathsf{c}_2), \mathsf{Succ}(\mathsf{c}_1, \mathsf{c}_2)\} \\
(I^*_P)^3_R = \quad & (I^*_P)^2_R \cup \{\mathsf{Desc}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{A}(\mathsf{c}_1)\} \\
(I^*_P)^4_R = \quad & (I^*_P)^3_R \cup \{\mathsf{R}(\mathsf{c}_1, \mathsf{c}_2), \mathsf{B}(\mathsf{c}_2), \mathsf{Desc}(\mathsf{c}_1, \mathsf{c}_2)\} \\
(I^*_P)^5_R = \quad & (I^*_P)^4_R \cup \{\mathsf{S}(\mathsf{c}_2, \mathsf{c}_2), \mathsf{T}(\mathsf{c}_2, \mathsf{c}_2), \mathsf{Succ}(\mathsf{c}_2, \mathsf{c}_2)\} \\
(I^*_P)^6_R = \quad & (I^*_P)^5_R \cup \{\mathsf{Desc}(\mathsf{c}_2, \mathsf{c}_2), \mathsf{A}(\mathsf{c}_2)\} \\
(I^*_P)^7_R = \quad & (I^*_P)^6_R \cup \{\mathsf{Cycle}, \mathsf{R}(\mathsf{c}_2, \mathsf{c}_1), \mathsf{B}(\mathsf{c}_1), \mathsf{Succ}(\mathsf{c}_2, \mathsf{c}_1)\}
\end{aligned}
$$

♢

4.1.3 Complexity of Reasoning for Positive Programs

We now pinpoint the computational costs for reasoning over programs that are MSA or MFA. To this end, we establish tight bounds w.r.t. both combined complexity for (un)bounded arity and data complexity. In particular, we focus on the problem of boolean conjunctive query answering; it is straightforward to see that the problem of

subsumption entailment can be polynomially reduced to fact entailment which is an instance of boolean conjunctive query answering. Table 4.1 provides a summary of the complexity results for reasoning over different types of logic programs. First, we examine the general case for which the input program is MSA or MFA but without requiring any further restriction on the shape of the rules.

Type of program for BCQ answering	**Data complexity**	**Combined complexity (un)bounded arity**
MFA/MSA program with unrestricted rules	P-complete (Theorem 36)	2ExpTime-complete (Theorem 36)
MFA/MSA program with $\exists$-1 rules	P-complete (Theorem 37)	ExpTime-complete (Theorem 37)

Table 4.1: Complexity of reasoning for decidable classes of positive existential rules

Theorem 36. *Let P be an MFA program, let F be a set of facts and let q be a BCQ. Deciding $P \cup F \models q$ is* 2ExpTime*-complete w.r.t. combined complexity for (un)bounded arity and* P*-complete w.r.t. data complexity. The complexity bounds are the same if P is MSA.*

Proof. (Membership) Let a, b, c, d and e be defined as in the proof of Proposition 23; by the same proposition, $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ is bounded by $b \cdot (c \cdot d)^{e \cdot a^d}$ which is doubly exponential in the size of P and F, even if e is fixed. For data complexity, the bound only depends on c and, thus, becomes polynomial. Moreover, testing whether there exists a homomorphism from q to $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ can be done in time polynomial in the size of $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ and exponential in the size of q which yields the required bounds. Finally, by Theorem 34, if P is MSA, then P is MFA, which implies that the bounds carry over when P is MSA.

(Hardness) By Theorem 34 and Theorem 55 (which appears later in Section 4.3), we have that if P is WA, then P is MSA and, also, P is MFA. Therefore, 2ExpTime-hardness follows by reducing the problem of BCQ answering over a WA set of rules (which is proved to be 2ExpTime-hard even for bounded arity [44]) to BCQ answering

over an MFA or MSA set of rules. P-hardness for data complexity is a consequence of the fact that checking $F \cup P \models \alpha$ for a set of facts F, a datalog program P and a ground atom α is P-hard when fixing the program P. Thus, P-hardness can be shown by reducing ground atom entailment over a set of datalog rules to BCQ answering over a set of MFA or MSA rules. □

As shown by the subsequent theorem, the complexity bounds drop by one exponential factor for ∃-1 programs. This result could be of practical importance because, as our use cases suggest, ∃-1 rules can be used to describe complex entities with sufficient precision. The proof is similar to the one of Theorem 27 for the membership result and to the one of Theorem 24 for the hardness result.

Theorem 37. *Let P be an MFA ∃-1 program, let F be a set of facts and let q be a BCQ. Deciding $P \cup F \models q$ is* ExpTime*-complete w.r.t. combined complexity and* P*-complete w.r.t. data complexity, even when the arity of the predicates is bounded. The complexity bounds are the same if P is MSA.*

Proof. (Membership) Similarly to the proof of Theorem 27, the set $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ is bounded by $b \cdot c^e \cdot d^{ed}$; therefore, the limit on the size of $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$ drops to single exponential in the size of P and F; moreover, for bounded arity e becomes constant but without reducing the size of $\mathsf{Chase}_{\mathsf{sk}(P)}(F)$. By Theorem 34, the same bound holds for MSA. Regarding data complexity, the restriction on the number of atoms remains polynomial, both for MSA and MFA.

(Hardness) In order to prove ExpTime-hardness we adjust the result by Calì et al. [44] used in the proof of Theorem 24 for the case of weakly acyclic set of ∃-1 rules. We adopt a Turing machine construction similar to the one used in the proof of Theorem 27, with the difference that the constructed program is now ∃-1 and weakly acyclic.

Lemma 38. *For a weakly acyclic set of ∃-1 rules P, a set of facts F and a Boolean conjunctive query q, determining $F \cup P \models q$ is* ExpTime*-hard w.r.t. combined complexity, even if the arity of the predicates is bounded.*

1. Initialisation: if $w = \alpha_0 \dots \alpha_m$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{next_k}(\mathsf{x_i}, \mathsf{x_{i+1}}) \rightarrow \mathsf{state}_{\mathsf{q}_0}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{symbol}_{\alpha_\mathsf{i}}(\mathsf{x}_0, \mathsf{x_i}) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}_{\mathsf{m}+1})$$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{y})$$

2. Transition rules: for all $\delta = \langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle \in \Delta$

$$\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{y}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{v}, \mathsf{v}') \rightarrow \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \quad \text{if } \mathsf{m} = l$$

$$\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{next_k}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{next_k}(\mathsf{v}, \mathsf{v}') \rightarrow \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \quad \text{if } \mathsf{m} = r$$

3. Inertia: for all $\alpha \in \Sigma$

$$\mathsf{next_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{nextt}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

$$\mathsf{next_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{nextt}(\mathsf{y}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

4. Acceptance: for each accepting state $\mathsf{q_a} \in Q$

$$\mathsf{state}_{\mathsf{q_a}}(\mathsf{v}) \rightarrow \mathsf{accept}$$

Figure 4.2: Program $P^{\mathsf{comp}}_{\mathcal{T},w}$ simulating computation of a DTM $\mathcal{T}$ over w; the program is the same as in Figure 4.1 modulo the acceptance rule

Proof. To prove ExpTime-hardness w.r.t. combined complexity, we show that for every deterministic Turing machine (DTM) $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$, word $w \in \Sigma^*$ and number $k \geq 1$, we can construct a WA $\exists$-1 program $P_{\mathcal{T},w,k}$ with a special propositional symbol accept and a set of facts F such that

$$\mathcal{T} \text{ accepts } w \text{ in time } 2^k \quad \Leftrightarrow \quad F \cup P_{\mathcal{T},w,k} \models \mathsf{accept} \tag{4.8}$$

Let $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$ be a DTM such that the transition function Δ consists of tuples of the form $\langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle$, where $\mathsf{q}, \mathsf{q}' \in Q$, $\alpha, \alpha' \in \Sigma$ and $\mathsf{m} \in \{l, r\}$ depending on whether the head moves left or right.

First, given $k \geq 1$, we construct a WA $\exists$-1 program P^{exp}_k of polynomial size,

which encodes a chain of 2^k elements. We use this exponentially long chain to model both the tape of cells and the timeline of instants. F contains the facts $\mathsf{s}_0(\mathsf{c}_0), \mathsf{s}_0(\mathsf{c}_1), \mathsf{next}_0(\mathsf{c}_0, \mathsf{c}_1), \mathsf{min}_0(\mathsf{c}_0)$ and P_k^{exp} contains the following rules for each $i \in \{0, \ldots, k-1\}$:

$$\mathsf{s_i}(\mathsf{x}) \rightarrow \exists \mathsf{y}_1, \mathsf{y}_2 . \ell_\mathsf{i}(\mathsf{x}, \mathsf{y}_1) \wedge \mathsf{r_i}(\mathsf{x}, \mathsf{y}_2)$$

$$\ell_\mathsf{i}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{s}_{\mathsf{i}+1}(\mathsf{x})$$

$$\mathsf{r_i}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{s}_{\mathsf{i}+1}(\mathsf{x})$$

$$\ell_\mathsf{i}(\mathsf{z}, \mathsf{x}_1) \wedge \mathsf{r_i}(\mathsf{z}, \mathsf{x}_2) \rightarrow \mathsf{next}_{\mathsf{i}+1}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{r_i}(\mathsf{z}_1, \mathsf{x}_1) \wedge \ell_\mathsf{i}(\mathsf{z}_2, \mathsf{x}_2) \wedge \mathsf{next_i}(\mathsf{z}_1, \mathsf{z}_2) \rightarrow \mathsf{next}_{\mathsf{i}+1}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{min_i}(\mathsf{z}) \wedge \ell_\mathsf{i}(\mathsf{z}, \mathsf{x}) \rightarrow \mathsf{min}_{\mathsf{i}+1}(\mathsf{x})$$

We extend P_k^{exp} with the following two rules, which transitively close the precedence relation that arises from the binary predicate $\mathsf{next_k}$ between the elements of the chain:

$$\mathsf{next_k}(\mathsf{x}_1, \mathsf{x}_2) \rightarrow \mathsf{nextt}(\mathsf{x}_1, \mathsf{x}_2)$$

$$\mathsf{nextt}(\mathsf{x}_1, \mathsf{z}) \wedge \mathsf{nextt}(\mathsf{z}, \mathsf{x}_2) \rightarrow \mathsf{nextt}(\mathsf{x}_1, \mathsf{x}_2)$$

Next, we build $P_{\mathcal{T},w}^{\mathsf{comp}}$ which simulates the computation of $\mathcal{T}$ on w. We use the following predicates for our encoding:

- $\mathsf{state_q}(\mathsf{v})$ for $\mathsf{q} \in Q$, when $\mathcal{T}$ is at state q at time v;
- $\mathsf{head}(\mathsf{v}, \mathsf{x})$, when the head of $\mathcal{T}$ is on cell x at time v;
- $\mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x})$ for $\alpha \in \Sigma$, when cell x contains symbol α at time v;
- accept, when the computation of $\mathcal{T}$ on w has reached an accepting state.

The rules of $P_{\mathcal{T},w}^{\mathsf{comp}}$ are shown in Figure 4.2; also, let $P_{\mathcal{T},w,k} = P_k^{\mathsf{exp}} \cup P_{\mathcal{T},w}^{\mathsf{comp}}$. One can easily check that $P_{\mathcal{T},w,k}$ is a WA $\exists$-1 program. Indeed, the program P_k^{exp} is WA by construction and its predicates do not occur in any rule head of $P_{\mathcal{T},w}^{\mathsf{comp}}$. Moreover, $P_{\mathcal{T},w}^{\mathsf{comp}}$ is WA since it does not contain existential quantifiers. Finally, the size of $P_{\mathcal{T},w,k}$ is polynomial in the size of $\mathcal{T}$, w and k and the predicates in $P_{\mathcal{T},w,k}$ are of

bounded arity which causes the lemma to hold even if the arity of predicates from P is bounded.

It is easy to see that 4.8 holds: $\mathcal{T}$ accepts w if and only if the computation ends in an accepting state $\mathsf{q_a}$. This holds if and only if $\mathsf{Chase}_{P_{\mathcal{T},w,k}}(F)$ contains the fact $\mathsf{state}_{\mathsf{q_a}}(\mathsf{t})$ for some instant t and some state $\mathsf{q_a} \in Q$ which is equivalent to $F \cup P_{\mathcal{T},w,k} \models \mathsf{accept}$. □

ExpTime-hardness is established by reducing BCQ answering over an WA ∃-1 program (which by Lemma 38 is ExpTime-hard even for bounded arity) to BCQ answering over an MSA ∃-1 program (by Theorem 55 if P is WA, then P is MSA); the proof is the same for MFA. The lower bound for data complexity is obtained by reducing fact entailment over datalog rules to BCQ answering over ∃-1 MSA or ∃-1 MFA rules: the former problem is known to be P-hard and the reduction is possible since every datalog program is both ∃-1 MSA and ∃-1 MFA. □

4.2 Acyclicity Conditions for Nonmonotonic Existential Rules

After introducing chase termination conditions for positive existential rules, we focus on acyclicity criteria that are suitable for rules with negative bodies. In particular, we present a condition that guarantees finiteness of the stable model for programs that consist of nonmonotonic existential rules, as opposed to Section 4.1 where we only examined rules without negation. Furthermore, the conditions that follow are based on an analysis of whether one rule *relies* on another, in the sense that it might be 'triggered' by the other rule's application. To this end, we formalise in Section 4.2.1 a notion of *positive reliance* between rules which we use in Section 4.2.2 to define R-acyclicity as a condition that ensures boundedness of stable models. We conclude with Section 4.2.3 where tight complexity bounds for reasoning over R-acyclic programs are established.

4.2.1 Introducing Positive Reliances

As recalled in Proposition 14, every stable model of a logic program can be obtained from a (possibly infinite) sequence of consecutive rule applications. Insights about the semantic properties of a program can thus be gained by analysing, for all pairs of rules r_1 and r_2, whether an application of r_1 can potentially enable a later application of r_2.

Definition 39 (Positive Reliance)**.** *Let r_1 and r_2 be two rules of the form* (2.1) *such that $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$; w.l.o.g. we assume that $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$. Rule r_2* positively relies *on r_1 (written $r_1 \xrightarrow{+} r_2$) if there exists a set of facts F that contains no skolem terms and a substitution θ such that:*

$$B_1^+\theta \subseteq F \qquad \text{(P1)} \qquad\qquad B_2^-\theta \cap (F \cup H_1\theta) = \emptyset \qquad \text{(P4)}$$

$$B_1^-\theta \cap F = \emptyset \qquad \text{(P2)} \qquad\qquad B_2^+\theta \not\subseteq F \qquad \text{(P5)}$$

$$B_2^+\theta \subseteq F \cup H_1\theta \qquad \text{(P3)} \qquad\qquad H_2\theta \not\subseteq F \cup H_1\theta \qquad \text{(P6)}$$

Thus, $r_1 \xrightarrow{+} r_2$ holds if there is a situation (defined by F) where r_1 is applicable (conditions (P1) and (P2)), r_2 is not applicable (condition (P5)) and applying r_1 allows r_2 to be applied ((P3) and (P4) indicate applicability of r_2 that depends on the execution of r_1) and to derive something new ((P6) requires r_2 to deduce a fresh atom).

Example 40. Consider rules r_1 and r_2 obtained from Example 16.

$$\bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom}(\mathsf{u}, \mathsf{v_i}) \wedge \mathsf{carbon}(\mathsf{v_1}) \wedge \mathsf{oxygen}(\mathsf{v_2}) \wedge \mathsf{hydrogen}(\mathsf{v_3}) \wedge \mathsf{singleBond}(\mathsf{v_1}, \mathsf{v_2}) \wedge \mathsf{singleBond}(\mathsf{v_2}, \mathsf{v_3}) \rightarrow \mathsf{organicHydroxy}(\mathsf{u}) \qquad (r_1)$$

$$\mathsf{organicHydroxy}(\mathsf{x_2}) \rightarrow \exists_{\mathsf{i}=7}^{9}\mathsf{y_i}. \bigwedge_{\mathsf{i}=7}^{9} \mathsf{hasAtom}(\mathsf{x_2}, \mathsf{y_i}) \wedge \mathsf{carbon}(\mathsf{y_7}) \wedge \mathsf{oxygen}(\mathsf{y_8}) \wedge \mathsf{hydrogen}(\mathsf{y_9}) \wedge \mathsf{singleBond}(\mathsf{y_7}, \mathsf{y_8}) \wedge \mathsf{singleBond}(\mathsf{y_8}, \mathsf{y_9}) \qquad (r_2)$$

We find that $r_1 \xrightarrow{+} r_2$ since there exist F and θ defined below that satisfy (P1)–(P6).

$$\begin{aligned} F =& \{\mathsf{hasAtom}(\mathsf{a}, \mathsf{b}_\mathsf{i})\}_{\mathsf{i}=1}^{3} \cup \{\mathsf{carbon}(\mathsf{b}_1), \mathsf{oxygen}(\mathsf{b}_2), \mathsf{hydrogen}(\mathsf{b}_3)\} \cup \\ & \{\mathsf{singleBond}(\mathsf{b}_1, \mathsf{b}_2), \mathsf{singleBond}(\mathsf{b}_2, \mathsf{b}_3)\} \\ \theta =& \{\mathsf{u} \mapsto \mathsf{a}, \vec{\mathsf{v}} \mapsto \vec{\mathsf{b}}, \mathsf{x}_2 \mapsto \mathsf{a}\} \end{aligned}$$

In contrast, $r_2 \xrightarrow{+} r_1$. Intuitively, r_1 can only derive facts that are already necessary to apply r_2 in the first place, thus violating (P6). More formally, suppose that $r_2 \xrightarrow{+} r_1$ could be shown using F' and θ'. By (P1) and (P6), F' and θ' need to satisfy $B_2^+\theta' \subseteq F'$ and $H_1\theta' \not\subseteq F' \cup H_2\theta'$ which imply $\theta'(\mathsf{x}_2) \neq \theta'(\mathsf{u})$. Additionally, by (P3) we have $B_1^+\theta' \subseteq F' \cup H_2\theta'$ which implies $\{\mathsf{hasAtom}(\mathsf{u}, \mathsf{v}_\mathsf{i})\theta'\}_{\mathsf{i}=1}^{3} \subseteq F'$ and, thus, $B_1^+\theta' \subseteq F'$. However, the latter violates condition (P5), which requires $B_1^+\theta' \not\subseteq F'$. ♢

Various previous works consider similar notions. The *activation* relation by Greco et al. [103] is most similar to Definition 39, but allows F to contain function terms to accommodate arbitrary logic programs with functions. Our stronger restriction is needed to show $r_2 \xrightarrow{+} r_1$ in Example 40. This illustrates how we can take advantage of the specific structure of existential rules to discard certain potential interactions. Another similar notion, which has also inspired our definition of positive reliance, is the *rule dependency*[1] by Baget et al. [15, 17] that is formulated only for positive rules; nonetheless, Baget et al. omit condition (P6) which is needed to show $r_2 \xrightarrow{+} r_1$ in Example 40. Along the same lines, Deutsch et al. independently suggested the $\prec$-stratification relation for pairs of rules which includes condition (P6) but does not cover negation either.

We prove that checking positive reliance for two rules is NP-complete. Similar results are shown by Deutsch et al. [71] and by Baget et al. [20] for rules without negation, whereas no computational analysis is provided for the activation relation. The complexity refers to the size of the two involved rules rather than to the size of the whole program: in practice, positive reliances can be checked efficiently by checking

[1] For our definitions we selected the term 'reliance' instead of 'dependency' (which one could claim is more intuitive) as in the context of databases a dependency usually refers to a rule [176].

the applicability of one of the rules to a linear number of facts. NP-hardness is shown by reducing the problem of checking the existence of a homomorphism from a set of atoms to another to testing a positive reliance.

Theorem 41. *Given rules r_1 and r_2, the problem of deciding whether $r_1 \xrightarrow{+} r_2$ is* NP*-complete.*

Proof. In order to show the complexity for deciding $r_1 \xrightarrow{+} r_2$, let $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$, $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$ and $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$.

(Membership) Assume that $r_1 \xrightarrow{+} r_2$. Then there exists a set of facts F and a substitution θ that satisfy the conditions of Definition 39. By conditions (P1) and (P3) we can assume w.l.o.g. that

$$F \subseteq (B_1^+ \cup B_2^+)\theta \tag{4.9}$$

Thus the size of θ is $\mathsf{Var}(B_1^+ \cup B_2^+)$ which, like the size of F, is polynomial in the size of r_1 and r_2. It is thus possible to guess F and θ and to verify all conditions of Definition 39 in polynomial time.

(Hardness) The problem of checking whether there exists a homomorphism from a set of atoms to another set of atoms is known to be NP-complete [188]. Therefore we reduce the problem of homomorphism checking to the problem of checking positive reliance. In order to do that, we assign to each instance of the homomorphism problem for two sets of atoms $Q \neq \emptyset$ and Q' a pair of rules r_1 and r_2, such that:

$$\exists \text{ homomorphism } h : Q \to Q' \quad \Leftrightarrow \quad r_1 \xrightarrow{+} r_2 \tag{4.10}$$

Let Q and Q' be two such sets of atoms with $\mathsf{Var}(Q) = \vec{\mathsf{x}}$ and $\mathsf{Var}(Q') = \vec{\mathsf{x}'}$. Rules r_1 and r_2 are defined as follows (recall that $\mathsf{Aster}(\ldots)$ is defined in the beginning of the chapter):

$$r_1 : \quad \mathsf{Start} \to \exists \mathsf{y}', \vec{\mathsf{x}'}.\mathsf{Aster}(\mathsf{y}', Q')$$

$$r_2 : \quad \mathsf{Aster}(\mathsf{y}, Q) \to \mathsf{Goal}$$

where Start and Goal are fresh nullary predicates and y and y' are distinct fresh variables. Clearly, the size of r_1 and r_2 is polynomial in the size of Q' and Q.

Consider a substitution $\sigma = \{\mathsf{y}' \mapsto \mathsf{c}_{\mathsf{y}'}\} \cup \{\mathsf{x}' \mapsto \mathsf{c}_{\mathsf{x}'} \mid \mathsf{x}' \in \vec{\mathsf{x}'}\}$, where $\mathsf{c}_{\mathsf{y}'}$ and $\vec{\mathsf{c}_{\mathsf{x}'}}$ are the constant and the vector of constants used to skolemise y' and $\vec{\mathsf{x}'}$, respectively.

First, we show $\Rightarrow$ of (4.10). Given a homomorphism $h : Q \to Q'$, let $F = \{\mathsf{Start}\}$, let $\tau = \{\mathsf{x} \mapsto h(\mathsf{x}) \mid \mathsf{x} \in \vec{\mathsf{x}}\} \cup \{\mathsf{y} \mapsto \mathsf{y}'\}$ and let $\theta = \sigma \circ \tau$. To show the claim, we check whether the conditions of positive reliance are satisfied:

(P1) $B_1^+\theta \subseteq F$ because $\{\mathsf{Start}\} \subseteq \{\mathsf{Start}\}$.

(P2) $B_1^-\theta \cap F = \emptyset$ because $B_1^- = \emptyset$.

(P3) $B_2^+\theta \subseteq F \cup H_1\theta$ by $\mathsf{Aster}(\mathsf{y}, Q)\theta \subseteq \mathsf{Aster}(\mathsf{c}_{\mathsf{y}'}, Q'\sigma)$, which is a consequence of $h(Q) \subseteq Q'$ and $\theta(\mathsf{y}) = \mathsf{c}_{\mathsf{y}'}$.

(P4) $B_2^-\theta \cap (F \cup H_1\theta) = \emptyset$ because $B_2^- = \emptyset$.

(P5) $B_2^+\theta \not\subseteq F$ because $F = \{\mathsf{Start}\}$ and there is no propositional atom in $B_2^+\theta$.

(P6) $H_2\theta \not\subseteq F \cup H_1\theta$ by $\{\mathsf{Goal}\} \not\subseteq \{\mathsf{Start}\} \cup \mathsf{Aster}(\mathsf{c}_{\mathsf{y}'}, Q'\sigma)$.

Now we show $\Leftarrow$ of (4.10). If $r_1 \xrightarrow{+} r_2$, then there exists a set of facts F and a substitution θ such that (P1)–(P6) hold. By conditions (P3) and (P6), there exists at least one atom $\alpha \in B_2^+$ and an atom $\alpha' \in H_1$, such that $\alpha\theta = \alpha'$. Since α and α' are of the form $\hat{\mathsf{P}}(\mathsf{y}, \vec{\mathsf{t}})$ and $\hat{\mathsf{P}}(\mathsf{c}_{\mathsf{y}'}, \vec{\mathsf{c}_{\mathsf{t}'}})$, we have $\theta(\mathsf{y}) = \mathsf{c}'_{\mathsf{y}}$. Additionally, since y occurs in every atom of B_2^+, $\theta(\mathsf{y}) = \mathsf{c}'_{\mathsf{y}}$ and c'_{y} is a skolem constant, we have $B_2^+\theta \cap F = \emptyset$. Now by $B_2^+\theta \cap F = \emptyset$ and (P3), we have $B_2^+\theta \subseteq H_1$. Thus, for every atom in B_2^+ of the form $\hat{\mathsf{P}}(\mathsf{y}, \vec{\mathsf{t}})$ there exists an atom of the form $\hat{\mathsf{P}}(\mathsf{c}_{\mathsf{y}'}, \vec{\mathsf{c}_{\mathsf{t}'}})$ such that $\hat{\mathsf{P}}(\mathsf{y}, \vec{\mathsf{t}})\theta = \hat{\mathsf{P}}(\mathsf{c}_{\mathsf{y}'}, \vec{\mathsf{c}_{\mathsf{t}'}})$. Since $\theta(\mathsf{y}) = \mathsf{c}_{\mathsf{y}'}$, for every atom $\mathsf{P}(\vec{\mathsf{t}}) \in Q$, there exists an atom $\mathsf{P}(\vec{\mathsf{c}_{\mathsf{t}'}}) \in Q'$, such that $\mathsf{P}(\vec{\mathsf{t}})\theta = \mathsf{P}(\vec{\mathsf{c}_{\mathsf{t}'}})$. So, the function $h = \sigma^{-1} \circ \theta$ is a homomorphism from Q to Q'.

This shows (4.10) and establishes the claimed hardness result. □

4.2.2 R-acyclicity

One can assume that if a finite program has an infinite stable model, some rule with an existential quantifier must be applicable an infinite number of times. This, however, requires that there is a cycle in rule reliances, motivating the following definition.

Definition 42 (R-Acyclic)**.** *A program P is* R-acyclic *if there is no cycle of positive reliances $r_1 \xrightarrow{+} \ldots \xrightarrow{+} r_n \xrightarrow{+} r_1$ that involves a rule with an existential quantifier.*

Example 43. Let $P = \{g_m, r_1, r_2, r_{ho}, r_{oc}, r_{mc}\}$, as specified in Example 16.

$$\mathsf{methanol}(\mathsf{x}_1) \rightarrow \exists_{\mathsf{i}=1}^{6}\mathsf{y}_\mathsf{i}.\mathsf{molecule}(\mathsf{x}_1) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x}_1, \mathsf{y}_\mathsf{i}) \wedge \mathsf{carbon}(\mathsf{y}_1) \wedge \mathsf{oxygen}(\mathsf{y}_2) \wedge \bigwedge_{\mathsf{i}=3}^{6} \mathsf{hydrogen}(\mathsf{y}_\mathsf{i}) \wedge \bigwedge_{\mathsf{i}=2}^{5} \mathsf{singleBond}(\mathsf{y}_1, \mathsf{y}_\mathsf{i}) \wedge \mathsf{singleBond}(\mathsf{y}_2, \mathsf{y}_6) \qquad (g_m)$$

$$\mathsf{organicHydroxy}(\mathsf{x}_2) \rightarrow \exists_{\mathsf{i}=7}^{9}\mathsf{y}_\mathsf{i}.\bigwedge_{\mathsf{i}=7}^{9} \mathsf{hasAtom}(\mathsf{x}_2, \mathsf{y}_\mathsf{i}) \wedge \mathsf{carbon}(\mathsf{y}_7) \wedge \mathsf{oxygen}(\mathsf{y}_8) \wedge \mathsf{hydrogen}(\mathsf{y}_9) \wedge \mathsf{singleBond}(\mathsf{y}_7, \mathsf{y}_8) \wedge \mathsf{singleBond}(\mathsf{y}_8, \mathsf{y}_9) \qquad (r_2)$$

$$\bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom}(\mathsf{u}, \mathsf{v}_\mathsf{i}) \wedge \mathsf{carbon}(\mathsf{v}_1) \wedge \mathsf{oxygen}(\mathsf{v}_2) \wedge \mathsf{hydrogen}(\mathsf{v}_3) \wedge \mathsf{singleBond}(\mathsf{v}_1, \mathsf{v}_2) \wedge \mathsf{singleBond}(\mathsf{v}_2, \mathsf{v}_3) \rightarrow \mathsf{organicHydroxy}(\mathsf{u}) \qquad (r_1)$$

$$\mathsf{hasAtom}(\mathsf{x}, \mathsf{z}) \wedge \mathsf{oxygen}(\mathsf{z}) \rightarrow \mathsf{hasOxygen}(\mathsf{x}) \qquad (r_{ho})$$

$$\mathsf{molecule}(\mathsf{x}) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}_1) \wedge \mathsf{carbon}(\mathsf{z}_1) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}_2) \wedge \mathsf{carbon}(\mathsf{z}_2) \wedge \mathsf{z}_1 \neq \mathsf{z}_2 \rightarrow \mathsf{multiCarbonMolecule}(\mathsf{x}) \qquad (r_{mc})$$

$$\mathsf{molecule}(\mathsf{x}) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}) \wedge \mathsf{carbon}(\mathsf{z}) \wedge \ \mathbf{not}\ \mathsf{multiCarbonMolecule}(\mathsf{x}) \rightarrow \mathsf{oneCarbonMolecule}(\mathsf{x}) \qquad (r_{oc})$$

The complete list of positive reliances appears in Figure 4.3; we thus infer that P is R-acyclic. ♢

Similarly to R-acyclicity, Baget et al. proposed the aGRD chase termination condition that characterises positive programs with an acyclic graph of rule dependencies [17]. On the other hand, Deutsch et al. used their $\prec$-stratification notion to split a positive program into a set of connected components and then apply weak acyclicity to each of the components [71].

Theorem 44. *Checking whether a program P is R-acyclic is* coNP*-complete.*

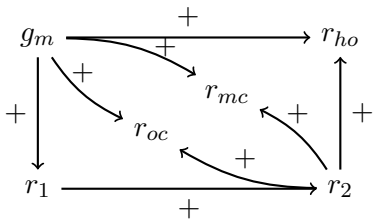


Figure 4.3: Positive reliances for program P of Example 43

Proof. Membership is a consequence of the fact that it can be checked in NP whether P is *not* R-acyclic. Indeed, if P is not R-acyclic, it has a cycle of length $n \leq |P|$. One can guess such a cycle $r_0 \xrightarrow{+} \ldots \xrightarrow{+} r_{n-1} \xrightarrow{+} r_0$ and justifications F_i, θ_i for each of the positive reliances $r_i \xrightarrow{+} r_{(i+1) \bmod n}$; these choices can be verified in polynomial time. For hardness, consider a rule $r_3 : \mathsf{Goal} \rightarrow \mathsf{Start}$ and rules r_1 and r_2 and atoms Q and Q' as specified in the hardness proof of Theorem 41. Clearly, the program $\{r_1, r_2, r_3\}$ is not R-acyclic if and only if there exists homomorphism $h : Q \rightarrow Q'$. □

Please note that similarly to the so-far acyclicity conditions, R-acyclicity does not recognise *only* programs with stable models of infinite size, that is it is possible for a program to be R-acyclic and have a finite stable model; Example 57 provides such a case.

4.2.3 Complexity of Reasoning for R-Acyclic Programs

The main result of this section shows that entailment under stable model semantics is decidable for R-acyclic programs. In particular, we prove that fact entailment for R-acyclic programs is coN2ExpTime-complete w.r.t. program complexity and coNP-complete w.r.t. data complexity; hardness for coN2ExpTime can be shown by reducing the word problem of 2ExpTime-bounded non-deterministic Turing machines to cautious entailment.

Theorem 45. *Let P be an R-acyclic program, let F be a set of facts and let α be a fact. Every stable model of $P \cup F$ has size doubly exponential in the size of P and polynomial in the size of F. Deciding $P \cup F \models \alpha$ is* coN2ExpTime*-complete w.r.t. program complexity and* coNP*-complete w.r.t. data complexity.*

The subsequent proof of this theorem involves a contradiction argument: we show that if a desired property does not hold, then there must be reliances that violate R-acyclicity. According to Definition 39, in order to show the existence of a reliance we need to define a suitable set of facts F and substitution θ, where F does not contain skolem terms. Facts that are obtained during a derivation usually do not have this property, but a suitable set of facts can still be obtained by replacing skolem terms with fresh constants. This replacement is used in proofs in this and the following two chapters, so we give a formal definition.

Definition 46. *For a set of facts F, the mapping γ_F on ground terms is recursively defined as follows:*

- *if* t *is a constant, let* $\gamma_F(\mathsf{t}) = \mathsf{t}$*;*
- *if* $\mathsf{t} \in \mathsf{Terms}(F) \setminus \mathsf{Const}(F)$*, let* $\gamma_F(\mathsf{t}) = \mathsf{c}_\mathsf{t}$ *be a unique fresh constant symbol;*
- *if* $\mathsf{t} = \mathsf{f}(\vec{\mathsf{s}}) \notin \mathsf{Terms}(F)$*, let* $\gamma_F(\mathsf{t}) = \mathsf{f}(\gamma_F(\vec{\mathsf{s}}))$.

We apply γ_F to (sets of) formulae and to substitutions by applying it to all terms in these structures.

Proof of Theorem 45. We first prove a lemma according to which the stable models of $P \cup F$ never contain any fact that uses a cyclic term; next we use this lemma to establish an upper bound on the number of distinct terms that such a model may contain.

Lemma 47. *For an R-acyclic program P and a set of facts F, every stable model of $P \cup F$ does not contain a cyclic term.*

Proof. Suppose for a contradiction that there is a stable model $\mathcal{M}$ of $P \cup F$ and a fact $\alpha \in \mathcal{M}$, such that $\mathsf{Terms}(\alpha)$ contains a cyclic term. By Proposition 14, we have $\mathcal{M} = T^\infty_{\mathsf{GL}(P,\mathcal{M})}(F)$. Thus, α is derived by some finite chain of applications of rules from $\mathsf{GL}(P, \mathcal{M})$ to F. Let $r_1, \ldots, r_m$ be a minimal sequence of rules $r_i \in \mathsf{GL}(P, \mathcal{M})$ that derive α, that is, $\alpha \in r_m(\ldots r_1(F)\ldots)$ and this property does not hold if any of the rules is omitted from the sequence. Let $F_1 = F$ and let

$F_{i+1} = r_i(F_i)$ for all $i \in \{1, \ldots, m-1\}$. Let $r'_i \in P$ be a rule and let θ_i be a substitution such that $r_i \in \mathsf{GL}(P, \mathcal{M})$ is obtained from $r'_i\theta_i$ by removing the negative body atoms. We show that, for every $i \in \{1, \ldots, m-1\}$, there is a reliance $r'_i \xrightarrow{+} r'_{i+1}$. Let $\mathsf{sk}(r'_i) = (B_i^+, B_i^-, H_i)$ and $\mathsf{sk}(r'_{i+1}) = (B_{i+1}^+, B_{i+1}^-, H_{i+1})$ and set $G = \gamma_{F_i}(F_i)$ and $\sigma = \gamma_{F_i}(\theta_i \cup \theta_{i+1})$ (it might be necessary to rename variables in r'_{i+1} to ensure $\mathsf{Var}(r'_i) \cap \mathsf{Var}(r'_{i+1}) = \emptyset$). We show that the conditions of Definition 39 are satisfied.

(P1) $B_i^+\sigma \subseteq G$ since $r_i = \mathsf{sk}(r'_i)\theta_i$ is applicable to F_i.

(P2) $B_i^-\sigma \cap G = \emptyset$ since $r_i = \mathsf{sk}(r'_i)\theta_i \in \mathsf{GL}(P, \mathcal{M})$.

(P3) $B_{i+1}^+\sigma \subseteq G \cup H_i\sigma$ since $G \cup H_i\sigma = \gamma_{F_i}(F_{i+1})$.

(P4) $B_{i+1}^-\sigma \cap (G \cup H_i\sigma) = \emptyset$ since $r_{i+1} = \mathsf{sk}(r'_{i+1})\theta_{i+1} \in \mathsf{GL}(P, \mathcal{M})$.

(P5) $B_{i+1}^+\sigma \not\subseteq G$ since otherwise r_i would not be required to derive α.

(P6) $H_{i+1}\sigma \not\subseteq G \cup H_i\sigma$ since otherwise r_{i+1} would not be required to derive α.

Thus, we obtain a chain of reliances $r'_1 \xrightarrow{+} \ldots \xrightarrow{+} r'_m$. Since F does not contain functional terms of the form $\mathsf{g}(\vec{\mathsf{s}})$, every such term that occurs in α must have been introduced by the application of some rule r_i. There is only one rule $r_\mathsf{g} \in P$ that can introduce a term of the form $\mathsf{g}(\vec{\mathsf{s}})$. Thus, if α contains a cyclic term that contains a subterm $\mathsf{f}(\vec{\mathsf{t}})$ such that f occurs in $\vec{\mathsf{t}}$, then there are two distinct indices $k < \ell \leq m$ such that $r'_k = r'_\ell = r_\mathsf{f}$. The reliances shown above thus yield a cycle $r'_k \xrightarrow{+} \ldots \xrightarrow{+} r'_{\ell-1} \xrightarrow{+} r'_\ell = r'_k$. Since r_f contains an existential quantifier, this shows that P is not R-acyclic, contradicting our assumptions. □

Now we compute an upper bound on the number of distinct atoms that a stable model of $P \cup F$ may contain. Similarly to the proof of Theorem 24, let c be the number of constants in $F \cup P$, let e be the maximal number of variables in any rule of P, assume that $\mathsf{sk}(P)$ contains d distinct function symbols and let $a > 1$ be an upper bound for the arity of function symbols. We have already shown in the proof of Theorem 24 that there are at most $c^{a^d} \cdot d^{a^d} = (c \cdot d)^{a^d}$ distinct terms that occur in any stable model of $P \cup F$ that does not contain a cyclic term, which by Lemma 47

is the case. Moreover, grounding P with $(c \cdot d)^{a^d}$ terms yields a maximal number of $((c \cdot d)^{a^d})^e = (c \cdot d)^{e \cdot a^d}$ propositional rules. The entailments of P agree with the entailments of this propositional program. Cautious entailment for propositional logic programs with negation can be decided in coNP [65], so entailment over P can be decided in coN2ExpTime. If the size of P is fixed, the bound on the number of grounded rules is of the form $k_1 \cdot c^{k_2}$, where k_1 and k_2 are constants. The size of the grounded program is thus polynomial in the size of F, so entailment can be decided in coNP w.r.t. data complexity.

The claimed coNP-hardness w.r.t. data complexity is an immediate consequence of the fact that entailment is coNP-hard w.r.t. data complexity for datalog with negation (i.e., rules without existential quantifiers or function symbols) [65]. Indeed, every datalog program with negation is clearly R-acyclic, since it does not contain existential quantifiers.

To prove coN2ExpTime-hardness w.r.t. program complexity, we show that for every non-deterministic Turing machine (NDTM) $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$, word $w \in \Sigma^*$ and number $k \geq 1$, we can construct an R-acyclic program $P_{\mathcal{T},w,k}$ with a special propositional symbol reject such that

$$\mathcal{T} \text{ accepts } w \text{ in time } 2^{2^k} \quad \Leftrightarrow \quad P_{\mathcal{T},w,k} \not\models \mathsf{reject}. \tag{$*$}$$

The proof adapts standard reduction techniques [26, 65, 46, 149].

Let $\mathcal{T} = \langle Q, \Sigma, \Delta, q_0 \rangle$ be an NDTM such that the transition relation Δ consists of tuples of the form $\langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle$, where $\mathsf{q}, \mathsf{q}' \in Q$, $\alpha, \alpha' \in P$ and $\mathsf{m} \in \{l, r\}$ depending on whether the head moves left or right.

First, given $k \geq 1$, we construct an R-acyclic program P_k^{dexp} of polynomial size, which encodes a chain of 2^{2^k} elements. We use this double-exponentially long chain to model both the tape of cells and the timeline of instants. Our construction follows Calì et al. [46]. P_k^{dexp} contains the facts $\mathsf{r}_0(\mathsf{c}_0), \mathsf{r}_0(\mathsf{c}_1), \mathsf{succ}_0(\mathsf{c}_0, \mathsf{c}_1), \mathsf{min}_0(\mathsf{c}_0), \mathsf{max}_0(\mathsf{c}_1)$ and the following rules for each $i \in \{0, \ldots, k-1\}$:

$$\mathsf{r_i}(\mathsf{x}) \wedge \mathsf{r_i}(\mathsf{y}) \rightarrow \exists \mathsf{z}.\mathsf{s_i}(\mathsf{x}, \mathsf{y}, \mathsf{z})$$

$$\mathsf{s_i}(\mathsf{x}, \mathsf{y}, \mathsf{z}) \rightarrow \mathsf{r_{i+1}}(\mathsf{z})$$

1. Initialisation: if $w = \alpha_0 \dots \alpha_m$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{succ_k}(\mathsf{x_i}, \mathsf{x_{i+1}}) \rightarrow \mathsf{state}_{\mathsf{q}_0}(\mathsf{x}_0) \wedge \bigwedge_{0 \le \mathsf{i} \le \mathsf{m}} \mathsf{symbol}_{\alpha_\mathsf{i}}(\mathsf{x}_0, \mathsf{x_i}) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}_{\mathsf{m}+1})$$

$$\mathsf{min_k}(\mathsf{x}_0) \wedge \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{x}) \wedge \mathsf{succ_k}(\mathsf{x}, \mathsf{y}) \rightarrow \mathsf{symbol}_{\square}(\mathsf{x}_0, \mathsf{y})$$

2. Transition rules: for all $\delta = \langle \mathsf{q}, \alpha, \mathsf{q}', \alpha', \mathsf{m} \rangle \in \Delta$ with

$$\Delta[\delta] = \{\epsilon \in \Delta \mid \epsilon = \langle \mathsf{q}, \alpha, \mathsf{q}_*, \alpha_*, \mathsf{m}_* \rangle, \epsilon \neq \delta\}$$

$$\begin{aligned} &\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \\ &\mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{succ_k}(\mathsf{y}, \mathsf{x}) \wedge \\ &\mathsf{succ_k}(\mathsf{v}, \mathsf{v}') \wedge \textstyle\bigwedge_{\epsilon \in \Delta[\delta]} \textbf{not } \mathsf{select}_\epsilon(\mathsf{v}) \quad \rightarrow \quad &&\mathsf{select}_\delta(\mathsf{v}) \wedge \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \\ &&&\mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \qquad \text{if } \mathsf{m} = l \end{aligned}$$

$$\begin{aligned} &\mathsf{state_q}(\mathsf{v}) \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \\ &\mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{x}) \wedge \mathsf{succ_k}(\mathsf{x}, \mathsf{y}) \wedge \\ &\mathsf{succ_k}(\mathsf{v}, \mathsf{v}') \wedge \textstyle\bigwedge_{\epsilon \in \Delta[\delta]} \textbf{not } \mathsf{select}_\epsilon(\mathsf{v}) \quad \rightarrow \quad &&\mathsf{select}_\delta(\mathsf{v}) \wedge \mathsf{state}_{\mathsf{q}'}(\mathsf{v}') \wedge \mathsf{head}(\mathsf{v}', \mathsf{y}) \wedge \\ &&&\mathsf{symbol}_{\alpha'}(\mathsf{v}', \mathsf{x}) \qquad \text{if } \mathsf{m} = r \end{aligned}$$

3. Inertia: for all $\alpha \in \Sigma$ and $\delta \in \Delta$

$$\mathsf{succ_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{succt}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

$$\mathsf{succ_k}(\mathsf{v}, \mathsf{v}') \wedge \mathsf{head}(\mathsf{v}, \mathsf{x}) \wedge \mathsf{succt}(\mathsf{y}, \mathsf{x}) \wedge \mathsf{symbol}_\alpha(\mathsf{v}, \mathsf{y}) \rightarrow \mathsf{symbol}_\alpha(\mathsf{v}', \mathsf{y})$$

4. Acceptance: for each accepting state $\mathsf{q_a} \in Q$

$$\mathsf{state}_{\mathsf{q_a}}(\mathsf{v}) \rightarrow \mathsf{accept}$$

Figure 4.4: Program $P^{\mathsf{comp}}_{\mathcal{T},w}$ simulating computation of an NDTM $\mathcal{T}$ over w; the program is the same as in Figure 4.1 apart from $\mathsf{next_k}$ that is used in place of $\mathsf{succ_k}$ and negation that is used in the transition rules to simulate the non-determinism of the Turing machine

$$\mathsf{s_i}(\mathsf{x}, \mathsf{y}, \mathsf{z}) \wedge \mathsf{s_i}(\mathsf{x}, \mathsf{y}', \mathsf{z}') \wedge \mathsf{succ_i}(\mathsf{y}, \mathsf{y}') \rightarrow \mathsf{succ}_{\mathsf{i}+1}(\mathsf{z}, \mathsf{z}')$$

$$\mathsf{s_i}(\mathsf{x}, \mathsf{y}, \mathsf{z}) \wedge \mathsf{s_i}(\mathsf{x}', \mathsf{y}', \mathsf{z}') \wedge \mathsf{max_i}(\mathsf{y}) \wedge \mathsf{min_i}(\mathsf{y}') \wedge \mathsf{succ_i}(\mathsf{x}, \mathsf{x}') \rightarrow \mathsf{succ}_{\mathsf{i}+1}(\mathsf{z}, \mathsf{z}')$$

$$\mathsf{min_i}(\mathsf{x}) \wedge \mathsf{s_i}(\mathsf{x},\mathsf{x},\mathsf{y}) \rightarrow \mathsf{min_{i+1}}(\mathsf{y})$$

$$\mathsf{max_i}(\mathsf{x}) \wedge \mathsf{s_i}(\mathsf{x},\mathsf{x},\mathsf{y}) \rightarrow \mathsf{max_{i+1}}(\mathsf{y})$$

We extend P_k^{dexp} with the following two rules, which transitively close the precedence relation that arises from the binary predicate $\mathsf{succ_k}$ between the elements of the chain:

$$\mathsf{succ_k}(\mathsf{x},\mathsf{y}) \rightarrow \mathsf{succt}(\mathsf{x},\mathsf{y})$$

$$\mathsf{succt}(\mathsf{x},\mathsf{y}) \wedge \mathsf{succt}(\mathsf{y},\mathsf{z}) \rightarrow \mathsf{succt}(\mathsf{x},\mathsf{z})$$

Next, we build $P_{\mathcal{T},w}^{\mathsf{comp}}$ which simulates the computation of $\mathcal{T}$ on w. We use the following predicates for our encoding:

- $\mathsf{state_q}(\mathsf{v})$ for $\mathsf{q} \in Q$, when $\mathcal{T}$ is at state q at time v;
- $\mathsf{head}(\mathsf{v},\mathsf{x})$, when the head of $\mathcal{T}$ is on cell x at time v;
- $\mathsf{symbol}_\alpha(\mathsf{v},\mathsf{x})$ for $\alpha \in \Sigma$, when cell x contains symbol α at time v;
- $\mathsf{select}_\delta(\mathsf{v})$ for $\delta \in \Delta$, when the transition δ is selected at time v;
- accept, when the computation of $\mathcal{T}$ on w has reached an accepting state.

The rules of $P_{\mathcal{T},w}^{\mathsf{comp}}$ are shown in Figure 4.4. The program $P_{\mathcal{T},w,k}$ is defined as follows

$$P_{\mathcal{T},w,k} = P_k^{\mathsf{dexp}} \cup P_{\mathcal{T},w}^{\mathsf{comp}} \cup \{\ \mathbf{not}\ \mathsf{accept} \rightarrow \mathsf{reject}\}$$

where the latter rule is added to ensure entailment of reject only when none of the computation branches reaches an accepting state. One can verify that $P_{\mathcal{T},w,k}$ is an R-acyclic program. Indeed, the program P_k^{dexp} is clearly acyclic by construction and its predicates do not occur in any rule head of $P_{\mathcal{T},w}^{\mathsf{comp}}$. Moreover, $P_{\mathcal{T},w}^{\mathsf{comp}}$ is R-acyclic since it does not contain existential quantifiers. The size of $P_{\mathcal{T},w,k}$ is polynomial in the size of $\mathcal{T}$, w and k.

It is easy to see that $(*)$ holds: $\mathcal{T}$ accepts w if and only if at least one branch of the computation tree contains in its leaf an accepting state q_a. This holds if and only if at least one of the stable models of $P_{\mathcal{T},w,k}$ contains a fact of the form $\mathsf{state_{q_a}}(\mathsf{t})$ for some instant t; due to the acceptance rule, the latter is true if and only if there exists one stable model of $P_{\mathcal{T},w,k}$ that does not contain reject, which on its turn is true if and only if $P_{\mathcal{T},w,k} \not\models \mathsf{reject}$. □

4.3 Acyclicity Zoo

In this section we compare acyclicity conditions for positive rules. Figure 4.5 depicts the relationship between many of the acyclicity conditions discussed so far w.r.t. checking costs and generality. We have only considered conditions for which (some) computational bounds are known in order to better illustrate the trade-off between complexity and expressive power. For this reason, conditions such as λ-restrictedness [89], ω-restrictedness [228] and Γ-acyclicity [103] are not included here, but a comparison between them can be found at the end of Section 3.3.1.3. The occurrences of MSA and MFA in Figure 4.5 refer to universal MSA and universal MFA, respectively. MFA^{BA} and $\text{MFA}^{\exists\text{-}1}$ stand for MFA programs whose predicates are of bounded arity and that consist of $\exists$-1 rules, respectively (similarly for MSA). MFA^{BA} and $\text{MFA}^{\exists\text{-}1}$ appear in Figure 4.5 in order to demonstrate the drop (or not) in complexity when checking for MFA in special cases (similarly for MSA).

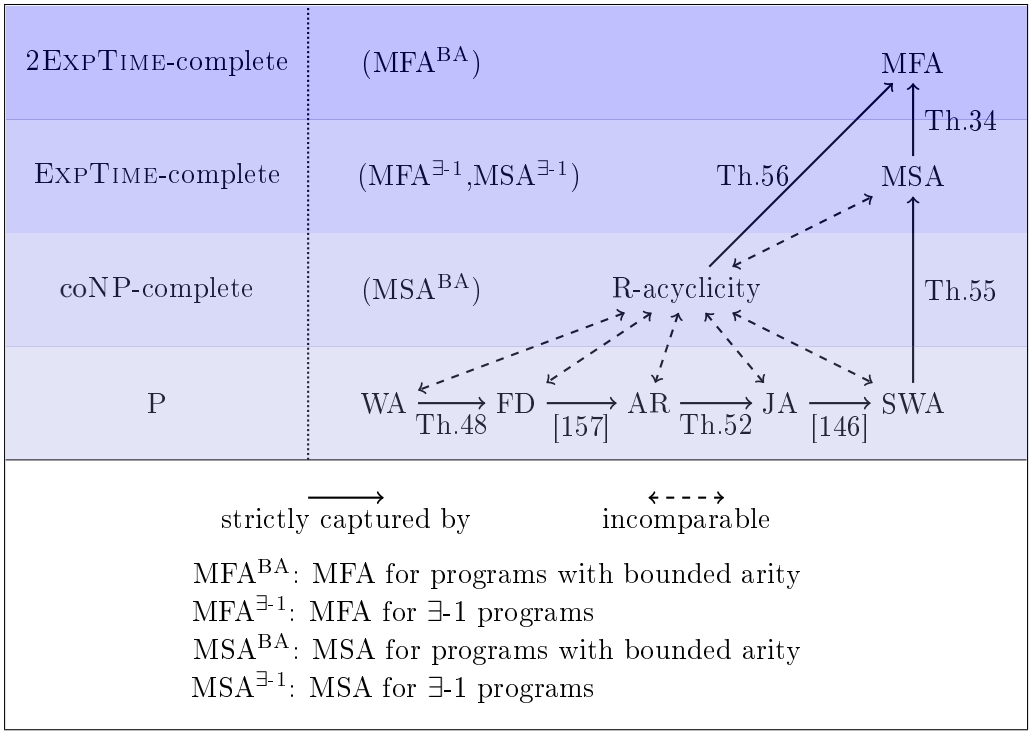


Figure 4.5: Comparison of acyclicity conditions w.r.t. (combined) complexity of checking and expressive power (FD, AR and R-acyclicity considered only for positive rules)

In this section, we compare R-acyclicity, FD and AR with other conditions only

in the context of positive rules. As a consequence, R-acyclicity can be viewed as a strengthened version of aGRD [17] or a weakened version of $\prec$-stratification [71]. As we mentioned earlier, R-acyclicity is stronger than aGRD due to the extra condition (P6) and the requirement for the cycle to involve an existential rule; it is also weaker than $\prec$-stratification because of the further weak acyclicity check that $\prec$-stratification specifies. Extending R-acyclicity in a way similar to $\prec$-stratification by performing an additional acyclicity check is possible and could be of interest for future work. Cuenca Grau et al. [99] provide some further insights about the relationship between $\prec$-stratification and enhanced versions of aGRD. For the purposes of our related work discussion, when we refer to R-acyclicity we imply the condition defined by Definition 42.

In the remainder of the section, we provide all the necessary proofs to support the relationships depicted in Figure 4.5.

Theorem 48. *If a set of positive rules P is WA, then P is FD.*

Proof. Let P be a set of rules that is not FD. Then, there exist a rule $r \in P$, an atom $\mathsf{q}(\vec{\mathsf{t}})$ in the head of r, a j-th term of $\vec{\mathsf{t}}$ equal to an existential variable y, and a variable $\mathsf{x} \in \vec{\mathsf{x}}$ such that each position $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_B(\mathsf{x})$ is P-recursive with $\mathsf{q}|_\mathsf{j}$. The set $\mathsf{Pos}_B(\mathsf{x})$ is not empty ($\vec{\mathsf{x}}$ contains precisely those variables occurring both in the body and the head of the rule), so let $\mathsf{p}|_\mathsf{i}$ be a position in $\mathsf{Pos}_B(\mathsf{x})$. The WA dependency graph $\mathsf{WAG}(P)$ then contains a special edge from $\mathsf{p}|_\mathsf{i}$ to $\mathsf{q}|_\mathsf{j}$. Furthermore, since $\mathsf{q}|_\mathsf{j}$ is P-recursive with $\mathsf{p}|_\mathsf{i}$, $\mathsf{WAG}(P)$ contains a cycle going through $\mathsf{p}|_\mathsf{i}$ and $\mathsf{q}|_\mathsf{j}$. Thus, $\mathsf{WAG}(P)$ clearly contains a cycle containing a special edge, so P is not WA. □

Theorem 48 in combination with the following Example show that WA is strictly subsumed by FD.

Example 49. Let P be the set containing rules r_1 and r_2.

$$\mathsf{R}(\mathsf{z},\mathsf{x}) \wedge \mathsf{A}(\mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{S}(\mathsf{x},\mathsf{y}) \qquad (r_1)$$

$$\mathsf{S}(\mathsf{x}_1,\mathsf{x}_2) \rightarrow \mathsf{R}(\mathsf{x}_1,\mathsf{x}_2) \qquad (r_2)$$

P is not WA since the WA dependency graph contains a special edge from $\mathsf{R}|_2$ to $\mathsf{S}|_2$ and a regular edge from $\mathsf{S}|_2$ to $\mathsf{R}|_2$. However, P is FD because position $\mathsf{S}|_2$ is not P-recursive with $\mathsf{A}|_1 \in \mathsf{Pos}_B(\mathsf{x})$. ♢

Regarding the next strict containment Lierler and Lifschitz [157] show that if a program is FD, then it is AR. Together with Example 50, we derive that FD is strictly captured by AR.

Example 50. Let P be the set consisting of the following rules:

$$\mathsf{A}(\mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{R}(\mathsf{x},\mathsf{y}) \tag{r_1}$$

$$\mathsf{R}(\mathsf{x}_1,\mathsf{x}_2) \rightarrow \mathsf{S}(\mathsf{x}_1,\mathsf{x}_2) \tag{r_2}$$

$$\mathsf{S}(\mathsf{z},\mathsf{x}) \wedge \mathsf{B}(\mathsf{x}) \rightarrow \mathsf{A}(\mathsf{x}) \tag{r_3}$$

$\mathsf{WAG}(P)$ contains a special edge from $\mathsf{A}|_1$ to $\mathsf{R}|_2$ as well as regular edges from $\mathsf{R}|_2$ to $\mathsf{S}|_2$ and from $\mathsf{S}|_2$ to $\mathsf{A}|_1$; thus, $\mathsf{R}|_2$ is P-recursive with $\mathsf{A}|_1$. Consequently, rule r_1 cannot satisfy the conditions in Definition 10, so we have $\mathsf{R}|_2 \notin \mathsf{Pos}_{FD}(P)$ and thus P is not FD. In contrast, P is AR, as evidenced by the following argument ranking:

$$\alpha = \{\mathsf{A}|_1 \mapsto 0,\ \mathsf{B}|_1 \mapsto 0,\ \mathsf{R}|_1 \mapsto 0,\ \mathsf{R}|_2 \mapsto 1,\ \mathsf{S}|_1 \mapsto 0,\ \mathsf{S}|_2 \mapsto 1\}$$

♢

We next show that JA is strictly more general than AR. Towards this goal, we first prove an auxiliary lemma that establishes a relationship between the set jMove from the definition of JA and an argument ranking; next, we use this lemma to prove that AR implies JA; moreover, we present Example 53 that shows this inclusion to be proper.

Lemma 51. *Let P be a set of positive rules, let α be an argument ranking for P, let y be an existentially quantified variable in a rule $r \in P$ and let $\mathsf{jMove}_P(r,\mathsf{y})$ be the set of positions used in the definition of JA. For each position $\mathsf{p}|_\mathsf{i} \in \mathsf{jMove}_P(r,\mathsf{y})$, position $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_H(\mathsf{y})$ exists such that $\alpha(\mathsf{p}|_\mathsf{i}) \geq \alpha(\mathsf{q}|_\mathsf{j})$.*

Proof. Let y be an existentially quantified variable occurring in some rule $r \in P$ and consider an arbitrary position $\mathsf{p}|_\mathsf{i} \in \mathsf{jMove}_P(r,\mathsf{y})$. We prove the claim by induction

on the definition of $\mathsf{jMove}_P(r, \mathsf{y})$. The base case when $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{y})$ is immediate. Assume now that $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{x})$ for some variable x occurring in a rule $r' \in P$ and that $\mathsf{Pos}_B(\mathsf{x}) \subseteq \mathsf{jMove}_P(r, \mathsf{y})$, so $\mathsf{p}|_\mathsf{i}$ needs to be added to $\mathsf{jMove}_P(r, \mathsf{y})$. By the definition of an argument ranking and since $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{x})$, position $\mathsf{p}'|_\ell \in \mathsf{Pos}_B(\mathsf{x})$ exists such that $\alpha(\mathsf{p}|_\mathsf{i}) \geq \alpha(\mathsf{p}'|_\ell)$. But then, since $\mathsf{p}'|_\ell \in \mathsf{Pos}_B(\mathsf{x}) \subseteq \mathsf{jMove}_P(r, \mathsf{y})$, by the induction hypothesis we have that position $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_H(\mathsf{y})$ exists such that $\alpha(\mathsf{p}'|_\ell) \geq \alpha(\mathsf{q}|_\mathsf{j})$. Thus, $\alpha(\mathsf{p}|_\mathsf{i}) \geq \alpha(\mathsf{q}|_\mathsf{j})$, as required. □

Theorem 52. *If a set of positive rules P is AR, then P is JA.*

Proof. Assume that P is AR and let α be an argument ranking for P. We next prove the following claim: for each edge in $\mathsf{JAG}(P)$ from a variable y_1 to a variable y_2 and for each position $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_H(\mathsf{y}_2)$, there exists a position $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{y}_1)$ such that $\alpha(\mathsf{p}|_\mathsf{i}) < \alpha(\mathsf{q}|_\mathsf{j})$. Consider an arbitrary edge from y_1 to y_2 in $\mathsf{JAG}(P)$ and an arbitrary position $\mathsf{q}|_\mathsf{j} \in \mathsf{Pos}_H(\mathsf{y}_2)$. By the definition of the JA dependency graph, the rule r that contains y_2 also contains a universally quantified variable x such that x occurs in the head of r and $\mathsf{Pos}_B(\mathsf{x}) \subseteq \mathsf{jMove}_P(r, \mathsf{y}_1)$. Since α is an argument ranking for P, position $\mathsf{p}'|_\ell \in \mathsf{Pos}_B(\mathsf{x})$ exists such that $\alpha(\mathsf{p}'|_\ell) < \alpha(\mathsf{q}|_\mathsf{j})$. Since $\mathsf{p}'|_\ell \in \mathsf{jMove}_P(r, \mathsf{y}_1)$, by Lemma 51 position $\mathsf{p}|_\mathsf{i} \in \mathsf{Pos}_H(\mathsf{y}_1)$ exists such that $\alpha(\mathsf{p}|_\mathsf{i}) \leq \alpha(\mathsf{p}'|_\ell)$. Thus, we have $\alpha(\mathsf{p}|_\mathsf{i}) < \alpha(\mathsf{q}|_\mathsf{j})$ and so our claim holds. But then, this claim implies that the JA dependency graph $\mathsf{JAG}(P)$ is acyclic and therefore P is JA. □

Example 53. Let P be the set consisting of the following rules:

$$\mathsf{R}(\mathsf{z}_1, \mathsf{x}_1) \rightarrow \exists \mathsf{y}_1.\mathsf{S}(\mathsf{x}_1, \mathsf{y}_1) \qquad (r_1)$$

$$\mathsf{R}(\mathsf{z}_2, \mathsf{x}_2) \rightarrow \exists \mathsf{y}_2.\mathsf{S}(\mathsf{y}_2, \mathsf{x}_2) \qquad (r_2)$$

$$\mathsf{S}(\mathsf{x}_3, \mathsf{x}_4) \rightarrow \mathsf{T}(\mathsf{x}_3, \mathsf{x}_4) \qquad (r_3)$$

$$\mathsf{T}(\mathsf{x}_5, \mathsf{x}_6) \wedge \mathsf{T}(\mathsf{x}_6, \mathsf{x}_5) \rightarrow \mathsf{R}(\mathsf{x}_5, \mathsf{x}_6) \qquad (r_4)$$

Let α be an argument ranking for P and let $n_1 = \alpha(\mathsf{R}|_2)$, $n_2 = \alpha(\mathsf{S}|_1)$, $n_3 = \alpha(\mathsf{S}|_2)$, $n_4 = \alpha(\mathsf{T}|_1)$ and $n_5 = \alpha(\mathsf{T}|_2)$. By r_1 and r_2, it is $n_1 < n_3$ and $n_1 < n_2$, respectively. Moreover, due to r_3, $n_2 \leq n_4$ and $n_3 \leq n_5$ hold. Finally, by r_4 we have

$n_4 \leq n_1$ or $n_5 \leq n_1$ which in combination with the inequalities above yield a contradiction. Therefore, such α cannot exist and P is not AR. In contrast, we have $\mathsf{jMove}_P(r_1, \mathsf{y}_1) = \{\mathsf{S}|_2, \mathsf{T}|_2\}$ and $\mathsf{jMove}_P(r_2, \mathsf{y}_2) = \{\mathsf{S}|_1, \mathsf{T}|_1\}$ and so P is JA. ♢

Krötzsch and Rudolph [146] show that if a program is JA, then it is SWA. In particular, the two conditions differ only if at least one rule contains a body atom in which at least one variable occurs more than once, as it is shown by the following example. As a consequence, JA is strictly contained in SWA.

Example 54. Let P be the set of the following rules:

$$\mathsf{A}(\mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{R}(\mathsf{x},\mathsf{y}) \wedge \mathsf{R}(\mathsf{y},\mathsf{x}) \wedge \mathsf{R}(\mathsf{x},\mathsf{x}) \qquad (r_1)$$

$$\mathsf{R}(\mathsf{x},\mathsf{x}) \rightarrow \mathsf{B}(\mathsf{x}) \qquad (r_2)$$

$$\mathsf{B}(\mathsf{x}) \rightarrow \mathsf{A}(\mathsf{x}) \qquad (r_3)$$

By $\mathsf{Move}_P(r_1, \mathsf{y}) = \{\langle \mathsf{R}(\mathsf{y},\mathsf{x}), 1\rangle, \langle \mathsf{R}(\mathsf{x},\mathsf{y}), 2\rangle\}$ we have that P is SWA; however, P is not JA because $\mathsf{jMove}_P(r_1, \mathsf{y}) = \{\mathsf{R}|_1, \mathsf{R}|_2, \mathsf{B}|_1, \mathsf{A}|_1\}$. ♢

Subsequently we show that MSA subsumes SWA which together with program P_b from Example 5 entail that SWA is property captured by MSA.

Theorem 55. *If a set of positive rules P is SWA, then P is universally MSA.*

Proof. Let $Q = \mathsf{MSA}(P)$, let $I^0, I^1, \ldots$ be the chase sequence of Q w.r.t. I^*_P and let I^∞ be the chase of Q w.r.t. I^*_P. Furthermore, let h be the extended substitution from $\mathsf{HU}(\mathsf{sk}(P))$ to $\mathsf{HU}(Q)$ such that $h(\mathsf{t}) = \mathsf{c}^\mathsf{i}_r$ if t is of the form $\mathsf{f}^\mathsf{i}_r(\ldots)$ and $h(\mathsf{t}) = \mathsf{t}$ if t is a constant.

We prove the following property (†): for each rule $r \in P$, each existentially quantified variable $\mathsf{y_i}$ occurring in r, each $\mathsf{p}(\vec{\mathsf{t}}) \in I^\infty$ where $\mathsf{p} \notin \{\mathsf{Succ}, \mathsf{Desc}, \mathsf{Cycle}\}$ if there exists $\mathsf{t_j} \in \vec{\mathsf{t}}$ such that $\mathsf{t_j} = \mathsf{c}^\mathsf{i}_r$, then a substitution σ and a place $\langle \alpha, j\rangle \in \mathsf{Move}_P(r, \mathsf{y_i})$ exist such that $\mathsf{p}(\vec{\mathsf{t}}) = h(\alpha\sigma)$. The proof is by induction on the length of the chase. Since $I^0 = I^*_P$ does not contain a constant of the form c^i_r, property (†) follows immediately for I^0. Assume now that property (†) holds for some I^{k-1} and consider an arbitrary rule $r \in P$, an existentially quantified variable $\mathsf{y_i}$ in r, a fact $\mathsf{p}(\vec{\mathsf{t}}) \in I^k \setminus I^{k-1}$

with $\mathsf{p} \notin \{\mathsf{Succ}, \mathsf{Desc}, \mathsf{Cycle}\}$ and a term $\mathsf{t}_\mathsf{j} \in \vec{\mathsf{t}}$ such that $\mathsf{t}_\mathsf{j} = \mathsf{c}^\mathsf{i}_r$. The fact $\mathsf{p}(\vec{\mathsf{t}})$ is derived in I^k from the head atom α_1 of some rule $r_1 \in Q$. Let σ be the substitution used in the rule application; clearly, we have $\alpha_1\sigma = \mathsf{p}(\vec{\mathsf{t}})$. Furthermore, let $r_2 \in P$ be the rule such that $r_1 = \mathsf{MSA}(r_2)$, let $r_3 = \mathsf{sk}(r_2)$ and let α_3 be the head atom of r_3 that corresponds to α_1; clearly, we have $\mathsf{p}(\vec{\mathsf{t}}) = h(\alpha_3\sigma)$. Now if α_1 contains c^i_r in position j, then $r = r_1$ since r_1 is the only rule that contains c^i_r; thus, $\langle \alpha_3, j\rangle \in \mathsf{Out}(r, \mathsf{y}_\mathsf{i}) \subseteq \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$, so property (†) holds. Otherwise, α contains at position j a universally quantified variable x such that $\sigma(\mathsf{x}) = \mathsf{c}^\mathsf{i}_r$. Let $\beta_1, \ldots, \beta_n$ be the body atoms of r_1 that contain x; clearly, $\{\beta_1\sigma, \ldots, \beta_n\sigma\} \subseteq I^{k-1}$. All these atoms satisfy the induction assumption, so for each $\beta \in \{\beta_1, \ldots, \beta_n\}$ and each ℓ such that variable x is the ℓ-th argument of β, a place $\langle \beta', \ell\rangle \in \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$ and substitution τ exist such that $\beta\sigma = h(\beta'\tau)$. Let σ' be the substitution obtained from σ by setting $\sigma'(\mathsf{v}) = \tau(\mathsf{v})$ for each variable v for which $\tau(\mathsf{v})$ is a functional term; clearly, $\beta\sigma' = \beta'\tau$. But then, $\mathsf{In}(r_1, \mathsf{x}) \sqsubseteq \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$; hence, by the definition of Move_P, we have that $\langle \alpha_3, j\rangle \in \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$, so property (†) holds.

We additionally prove that if $\mathsf{Succ}(\mathsf{c}^\mathsf{i}_r, \mathsf{c}^{\mathsf{i}'}_{r'}) \in I^\infty$ for some i and i', then $r \rightsquigarrow_P r'$. Consider an arbitrary such fact, let y_i be the existentially quantified variable of r corresponding to c^i_r and let k be the smallest integer such that $\mathsf{Succ}(\mathsf{c}^\mathsf{i}_r, \mathsf{c}^{\mathsf{i}'}_{r'}) \in I^k$. Clearly, $\mathsf{Succ}(\mathsf{c}^\mathsf{i}_r, \mathsf{c}^{\mathsf{i}'}_{r'})$ is derived in I^k from the head atom $\mathsf{Succ}(\mathsf{x}, \mathsf{c}^{\mathsf{i}'}_{r'})$ of rule r', where $\mathsf{x} \in \mathsf{fr}(r')$. Let σ be the substitution used in the rule application; thus, $\sigma(\mathsf{x}) = \mathsf{c}^\mathsf{i}_r$. Let $\beta_1, \ldots, \beta_n$ be the body atoms of r' that contain x; clearly, we have $\{\beta_1\sigma, \ldots, \beta_n\sigma\} \subseteq I^{k-1}$. All these atoms satisfy property (†), so for each $\beta \in \{\beta_1, \ldots, \beta_n\}$ and for each variable x that is the ℓ-th argument of β, a place $\langle \beta', \ell\rangle \in \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$ and substitution τ exist such that $\beta\sigma = h(\beta'\tau)$. But then, as in the previous paragraph we can take substitution $\sigma' = \sigma \circ h^{-1}$ and have $\beta\sigma' = \beta\tau$ which yields $\mathsf{In}(r', \mathsf{x}) \sqsubseteq \mathsf{Move}_P(r, \mathsf{y}_\mathsf{i})$ and, thus, $r \rightsquigarrow_P r'$.

Assume now that P is not MSA. As a consequence $\mathsf{Cycle} \in I^\infty$ and therefore there exist t and t' such that $\{\mathsf{N}^\mathsf{i}_r(\mathsf{t}), \mathsf{Desc}(\mathsf{t}, \mathsf{t}'), \mathsf{N}^\mathsf{i}_r(\mathsf{t}')\} \subseteq I^\infty$ holds for some N^i_r due to rule (4.3). But then, since predicate N^i_r occurs in Q only in an atom $\mathsf{N}^\mathsf{i}_r(\mathsf{c}^\mathsf{i}_r)$, we have $\mathsf{t} = \mathsf{t}' = \mathsf{c}^\mathsf{i}_r$. Finally, since Desc is axiomatised in Q as the transitive closure of Succ

and by the claim proved in the previous paragraph, we derive $r \rightsquigarrow_P r$ and so P is not SWA. □

We now proceed to showing that MFA strictly generalises R-acyclicity.

Theorem 56. *If a set of positive rules P is R-acyclic, then P is universally MFA.*

Proof. Let P be a program that is not universally MFA. Then, the chase of P w.r.t. I^*_P contains a cyclic term, which by Lemma 47 implies that P is not R-acyclic. □

The next example shows why R-acyclicity does not generalise weak acyclicity.

Example 57. Let $P = \{r_1, r_2, r_3\}$ where r_1, r_2 and r_3 are defined next.

$$\mathsf{A}(\mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{R}(\mathsf{x},\mathsf{y}) \qquad (r_1)$$

$$\mathsf{R}(\mathsf{x}_1,\mathsf{x}_2) \rightarrow \mathsf{S}(\mathsf{x}_1,\mathsf{x}_2) \qquad (r_2)$$

$$\mathsf{S}(\mathsf{x}_3,\mathsf{z}) \wedge \mathsf{B}(\mathsf{z}) \rightarrow \mathsf{A}(\mathsf{x}_3) \qquad (r_3)$$

The program P is WA as inferred by $\mathsf{WAG}(P)$ that appears in Figure 4.6. However it is not R-acyclic, since $r_1 \xrightarrow{+} r_2$, $r_2 \xrightarrow{+} r_3$ and $r_3 \xrightarrow{+} r_1$ as witnessed by the following sets of facts and substitutions for each pair of rules, respectively (where f is the function symbol used to skolemise y).

$$F_1 = \{\mathsf{A}(\mathsf{a})\} \qquad \theta_1 = \{\mathsf{x} \mapsto \mathsf{a}, \mathsf{x}_1 \mapsto \mathsf{a}, \mathsf{x}_2 \mapsto \mathsf{f}(\mathsf{a})\}$$

$$F_2 = \{\mathsf{R}(\mathsf{a},\mathsf{b}), \mathsf{B}(\mathsf{b})\} \qquad \theta_2 = \{\mathsf{x}_1 \mapsto \mathsf{a}, \mathsf{x}_2 \mapsto \mathsf{b}, \mathsf{x}_3 \mapsto \mathsf{a}, \mathsf{z} \mapsto \mathsf{b}\}$$

$$F_3 = \{\mathsf{S}(\mathsf{a},\mathsf{b}), \mathsf{B}(\mathsf{b})\} \qquad \theta_3 = \{\mathsf{x}_3 \mapsto \mathsf{a}, \mathsf{z} \mapsto \mathsf{b}, \mathsf{x} \mapsto \mathsf{a}\}$$

♢

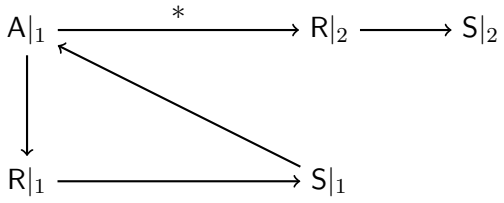


Figure 4.6: $\mathsf{WAG}(P)$ for program P of Example 57

By the containment relationships depicted in Figure 4.5, we derive that P from Example 57 is also FD, AR, JA, SWA, MSA and MFA. Therefore, Example 57 and Theorem 56 justify the strict containment between R-acyclicity and MFA. Moreover, Example 57 also provides a case that is not R-acyclic but WA, FD, AR, JA, SWA and MSA. Therefore, we only need an example that is R-acyclic but neither WA, nor FD, nor AR, nor JA, nor SWA, nor MSA to establish the incomparability of MSA with WA, FD, AR, JA, SWA and MSA. This is given next.

Example 58. Let P be the set consisting of the following rules:

$$\mathsf{R}(\mathsf{x}_1, \mathsf{x}_1) \wedge \mathsf{U}(\mathsf{x}_1, \mathsf{z}) \wedge \mathsf{U}(\mathsf{x}_2, \mathsf{z}) \rightarrow \mathsf{R}(\mathsf{x}_1, \mathsf{x}_2) \tag{r_1}$$

$$\mathsf{R}(\mathsf{z}, \mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{T}(\mathsf{x}, \mathsf{y}) \tag{r_2}$$

$$\mathsf{T}(\mathsf{z}, \mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{U}(\mathsf{x}, \mathsf{y}) \tag{r_3}$$

It is obvious that $r_1 \xrightarrow{+} r_2$, $r_1 \not\xrightarrow{+} r_3$, $r_2 \not\xrightarrow{+} r_1$, $r_2 \not\xrightarrow{+} r_2$, $r_2 \xrightarrow{+} r_3$, $r_3 \not\xrightarrow{+} r_2$, and $r_3 \not\xrightarrow{+} r_3$. We next argue that $r_3 \not\xrightarrow{+} r_1$, which implies that P is R-acyclic.

To see that $r_3 \not\xrightarrow{+} r_1$, assume that r_3 is applicable to a set of facts F and that the rule application derives a fact of the form $\mathsf{U}(\mathsf{a}, \mathsf{f}(\mathsf{a}))$. Now let $F' = F \cup \{\mathsf{U}(\mathsf{a}, \mathsf{f}(\mathsf{a}))\}$ and assume that a substitution σ' exists that makes r_1 applicable to F' but not to F; this rule application must 'use' the fact $\mathsf{U}(\mathsf{a}, \mathsf{f}(\mathsf{a}))$, which implies $\sigma'(\mathsf{x}_1) = \sigma'(\mathsf{x}_2) = \mathsf{a}$ and $\sigma'(\mathsf{z}) = \mathsf{f}(\mathsf{a})$. Furthermore, rule r_1 is applicable only if $\mathsf{R}(\mathsf{a}, \mathsf{a}) \in F$; but then, the rule application does not derive 'something new' since $\mathsf{R}(\mathsf{x}_1, \mathsf{x}_2)\sigma' = \mathsf{R}(\mathsf{a}, \mathsf{a})$. Consequently, we have $r_3 \not\xrightarrow{+} r_1$.

Let $Q = \mathsf{MSA}(P)$ and consider the chase of Q w.r.t. I^*_P, where $\mathsf{N}^2_{r_2}$, $\mathsf{N}^3_{r_3}$, $\mathsf{c}^2_{r_2}$ and $\mathsf{c}^3_{r_3}$ are the predicates and constants found in $\mathsf{MSA}(r_2)$ and $\mathsf{MSA}(r_3)$.

$$\begin{aligned}
(I^*_P)^0_Q &= \quad I^*_P \quad = \{\mathsf{R}(*,*), \mathsf{U}(*,*), \mathsf{T}(*,*)\} \\
(I^*_P)^1_Q &= (I^*_P)^0_Q \cup \{\mathsf{T}(*, \mathsf{c}_{\mathsf{r}_2}), \mathsf{Succ}(*, \mathsf{c}_{\mathsf{r}_2}), \mathsf{N}^2_{\mathsf{r}_2}(\mathsf{c}_{\mathsf{r}_2}), \mathsf{U}(*, \mathsf{c}_{\mathsf{r}_3}), \mathsf{Succ}(*, \mathsf{c}_{\mathsf{r}_3}), \mathsf{N}_{\mathsf{y}_3}(\mathsf{c}_{\mathsf{r}_3})\} \\
(I^*_P)^2_Q &= (I^*_P)^1_Q \cup \{\mathsf{U}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_3}), \mathsf{Succ}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_3}), \mathsf{Desc}(*, \mathsf{c}_{\mathsf{r}_2}), \mathsf{Desc}(*, \mathsf{c}_{\mathsf{r}_3})\} \\
(I^*_P)^3_Q &= (I^*_P)^2_Q \cup \{\mathsf{R}(*, \mathsf{c}_{\mathsf{r}_2}), \mathsf{Desc}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_3})\} \\
(I^*_P)^4_Q &= (I^*_P)^3_Q \cup \{\mathsf{T}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_2}), \mathsf{Succ}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_2})\} \\
(I^*_P)^5_Q &= (I^*_P)^4_Q \cup \{\mathsf{Desc}(\mathsf{c}_{\mathsf{r}_2}, \mathsf{c}_{\mathsf{r}_2})\}
\end{aligned}$$

$(I^*_P)^6_Q = (I^*_P)^5_Q \cup \{\mathsf{Cycle}\}$

The chase result contains Cycle, so P is not MSA. ♢

Chapter 5

Stable Model Uniqueness

In this chapter, we introduce a novel stratification condition called R-stratification that ensures stable model uniqueness and we prove that R-stratification extends the classical notion of stratification. Moreover, we define a deterministic procedure for computing the stable model of R-stratified programs which yields a semi-decision procedure for fact entailment. If an R-stratified program is also R-acyclic, this procedure results to an algorithm that is a promising candidate for efficiently reasoning over R-acyclic, R-stratified programs, such as the ones that arise from the previously presented chemistry examples. Part of the results presented in this chapter are published in [170].

The remainder of the chapter is organised as follows. In Section 5.1, we define *negative reliances*—a relation that captures when a rule can prevent the execution of another rule; we use negative reliances to define R-stratification. Then, in Section 5.2, we prove that R-stratified programs have at most one stable model and describe how one can compute this stable model. Next, in Section 5.3, we establish tight complexity bounds for reasoning over R-acyclic, R-stratified programs. In Section 5.4 we revisit various previously proposed stratification conditions and we conclude in Section 5.4 with a discussion on related work for nonmonotonic rules.

5.1 Negative Reliances and R-Stratification

While positive reliances defined in Chapter 4 allow us to estimate if one rule can 'trigger' another rule, the use of nonmonotonic negation may also give rise to the opposite interaction where one rule 'inhibits' another. In this section, we formalise this by defining *negative reliances* between rules. This suggests a new kind of *stratification*, which generalises the classical notion but can still be decided efficiently. This new kind of stratification emerges from the existence of negative reliances between rules and suggests an order of rule application towards computing the stable model. The impact of the new stratification condition on the existence of unique stable models is discussed in Section 5.2.

Definition 59 (Negative Reliance)**.** *Let r_1 and r_2 be two rules of the form* (2.1) *such that* $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ *and* $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$*; w.l.o.g. we assume that* $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$*. Rule* r_2 negatively relies *on* r_1 *(written* $r_1 \dashrightarrow r_2$*) if there exists a set of facts* F *that contains no skolem terms and a substitution* θ *such that:*

$$B_1^+\theta \subseteq F \qquad \text{(N1)} \qquad\qquad B_2^-\theta \cap H_1\theta \neq \emptyset \qquad \text{(N4)}$$

$$B_1^-\theta \cap F = \emptyset \qquad \text{(N2)} \qquad\qquad B_2^-\theta \cap F = \emptyset \qquad \text{(N5)}$$

$$B_2^+\theta \subseteq F \qquad \text{(N3)}$$

Example 60. Consider rules r_{oh} and g_{oh} from Example 16.

$$\begin{aligned}
&\bigwedge_{\mathsf{i}=1}^{3} \mathsf{hasAtom}(\mathsf{u}, \mathsf{v_i}) \wedge \mathsf{carbon}(\mathsf{v}_1) \wedge \\
&\qquad \mathsf{oxygen}(\mathsf{v}_2) \wedge \mathsf{hydrogen}(\mathsf{v}_3) \wedge \\
&\mathsf{singleBond}(\mathsf{v}_1, \mathsf{v}_2) \wedge \mathsf{singleBond}(\mathsf{v}_2, \mathsf{v}_3) \wedge \\
&\mathbf{not}\ \mathsf{g_{oh}}(\mathsf{v}_1) \wedge\ \mathbf{not}\ \mathsf{g_{oh}}(\mathsf{v}_2) \wedge\ \mathbf{not}\ \mathsf{g_{oh}}(\mathsf{v}_3) \rightarrow \mathsf{organicHydroxy}(\mathsf{u}) \wedge \mathsf{r_{oh}}(\mathsf{u}) \qquad (r_{oh}) \\
&\mathsf{organicHydroxy}(\mathsf{x}_2) \wedge\ \mathbf{not}\ \mathsf{r_{oh}}(\mathsf{x}_2) \rightarrow \exists_{\mathsf{i}=7}^{9}\mathsf{y_i}. \bigwedge_{\mathsf{i}=7}^{9} \mathsf{hasAtom}(\mathsf{x}_2, \mathsf{y_i}) \wedge \mathsf{carbon}(\mathsf{y}_7) \wedge \\
&\qquad\qquad \mathsf{oxygen}(\mathsf{y}_8) \wedge \mathsf{hydrogen}(\mathsf{y}_9) \wedge \\
&\qquad\qquad \mathsf{singleBond}(\mathsf{y}_7, \mathsf{y}_8) \wedge \mathsf{singleBond}(\mathsf{y}_8, \mathsf{y}_9) \wedge \\
&\qquad\qquad \mathsf{g_{oh}}(\mathsf{y}_7) \wedge \mathsf{g_{oh}}(\mathsf{y}_8) \wedge \mathsf{g_{oh}}(\mathsf{y}_9) \qquad (g_{oh})
\end{aligned}$$

We can show $r_{oh} \xrightarrow{-} g_{oh}$ using

$$\begin{aligned} F =& \{\mathsf{hasAtom}(\mathsf{a}, \mathsf{b_i})\}_{\mathsf{i}=1}^{3} \cup \{\mathsf{carbon}(\mathsf{b}_1), \mathsf{oxygen}(\mathsf{b}_2), \mathsf{hydrogen}(\mathsf{b}_3)\} \cup \\ & \{\mathsf{singleBond}(\mathsf{b}_1, \mathsf{b}_2), \mathsf{singleBond}(\mathsf{b}_2, \mathsf{b}_3), \mathsf{organicHydroxy}(\mathsf{a})\} \\ \theta =& \{\mathsf{u} \mapsto \mathsf{a}, \vec{\mathsf{v}} \mapsto \vec{\mathsf{b}}, \mathsf{x}_2 \mapsto \mathsf{a}\} \end{aligned}$$

Conversely, $g_{oh} \xrightarrow{\not -} r_{oh}$ follows from conditions (N3) and (N5) and the fact that F is not allowed to contain skolem terms. ♢

The following definition is inspired by the classical notion of stratification in logic programming. It uses the notion of positive reliances from Definition 39.

Definition 61 (R-Stratification)**.** *Given a program P, a sequence of disjoint programs $\vec{\mathsf{P}} = P_1, \ldots, P_n$ is an* R-stratification *of P if $P = \bigcup_{i=1}^{n} P_i$ and, for every two programs $P_i, P_j \in \vec{\mathsf{P}}$ and rules $r_1 \in P_i$ and $r_2 \in P_j$, we have:*

$$\textit{if } r_1 \xrightarrow{+} r_2 \textit{ then } i \leq j \qquad \textit{and} \qquad \textit{if } r_1 \xrightarrow{-} r_2 \textit{ then } i < j.$$

P is R-stratified *if it has an R-stratification.*

Example 62. For P consisting of rules r_{oh} and g_{oh} that appear above and rules g_m, r_{ho}, r_{mc} and r_{oc} that appear below we obtain the reliances depicted in Figure 5.1.

$$\begin{aligned} \mathsf{methanol}(\mathsf{x}_1) \rightarrow & \exists_{\mathsf{i}=1}^{6}\mathsf{y_i}.\mathsf{molecule}(\mathsf{x}_1) \wedge \bigwedge_{\mathsf{i}=1}^{6} \mathsf{hasAtom}(\mathsf{x}_1, \mathsf{y_i}) \wedge \mathsf{carbon}(\mathsf{y}_1) \wedge \\ & \mathsf{oxygen}(\mathsf{y}_2) \wedge \bigwedge_{\mathsf{i}=3}^{6} \mathsf{hydrogen}(\mathsf{y_i}) \wedge \\ & \bigwedge_{\mathsf{i}=2}^{5} \mathsf{singleBond}(\mathsf{y}_1, \mathsf{y_i}) \wedge \mathsf{singleBond}(\mathsf{y}_2, \mathsf{y}_6) && (g_m) \\ & \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}) \wedge \mathsf{oxygen}(\mathsf{z}) \rightarrow \mathsf{hasOxygen}(\mathsf{x}) && (r_{ho}) \\ \mathsf{molecule}(\mathsf{x}) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}_1) \wedge \mathsf{carbon}(\mathsf{z}_1) \wedge & \\ \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}_2) \wedge \mathsf{carbon}(\mathsf{z}_2) \wedge \mathsf{z}_1 \neq \mathsf{z}_2 \rightarrow & \mathsf{multiCarbonMolecule}(\mathsf{x}) && (r_{mc}) \\ \mathsf{molecule}(\mathsf{x}) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{z}) \wedge & \\ \mathsf{carbon}(\mathsf{z}) \wedge \ \mathbf{not}\ \mathsf{multiCarbonMolecule}(\mathsf{x}) \rightarrow & \mathsf{oneCarbonMolecule}(\mathsf{x}) && (r_{oc}) \end{aligned}$$

An R-stratification of P is therefore given by $P_1 = \{g_m, r_{oh}\}$, $P_2 = \{r_{ho}, g_{oh}, r_{mc}\}$

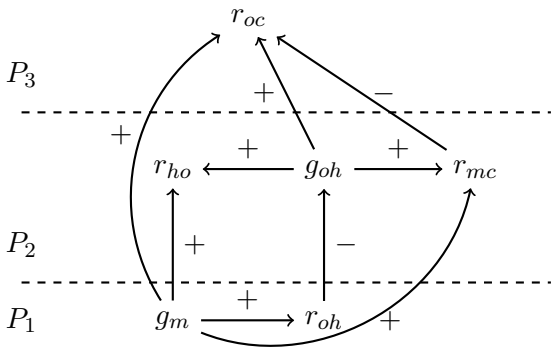


Figure 5.1: Positive and negative reliances for program P of Example 62

and $P_3 = \{r_{oc}\}$. In contrast, P is not stratified (according to Definition 12) due to predicates $\mathsf{g_{oh}}$ and $\mathsf{r_{oh}}$ in rules r_{oh} and g_{oh}. ♢

Together with the previous example, the next result shows that R-stratification properly generalises stratification.

Proposition 63. *If P is stratified, then P is R-stratified.*

Proof. Let $\vec{\mathsf{P}}$ be a stratification of $P = P_1, \ldots, P_n$. We show that $\vec{\mathsf{P}}$ is an R-stratification of P. Clearly, $P = \bigcup_{i=1}^{n} P_i$. Now consider arbitrary programs $P_i, P_j \in \vec{\mathsf{P}}$ and rules $r_1 \in P_i$ and $r_2 \in P_j$. If $r_1 \xrightarrow{+} r_2$ then, by (P3) and (P5), there is a predicate in the head of r_1 that occurs in a positive body atom of r_2. Therefore $i \leq j$, since $\vec{\mathsf{P}}$ is a stratification. Similarly, if $r_1 \xrightarrow{-} r_2$, (N4) implies that there is a predicate in the head of r_1 that occurs in a negative body atom of r_2 and thus $i < j$. □

The graph structure that is induced by reliances, defined next, can be used to decide R-stratification in practice, as shown in Proposition 65 below.

Definition 64 (Graph of Reliances)**.** *For a program P, the* graph of reliances $\mathsf{GoR}(P)$ *is a directed graph that has the rules of P as its vertices and two sets of edges:* positive edges *that correspond to the positive reliances of P and* negative edges *that correspond to the negative reliances of P.*

For example, Figure 5.1 illustrates the graph of reliances for program P of Example 62.

Proposition 65. *P is R-stratified if and only if its graph of reliances $\mathsf{GoR}(P)$ contains no directed cycle with a negative edge.*

Proof. Consider a program P. First, assume that $\mathsf{GoR}(P)$ has no cycles with negative edges. Let $\approx$ be an equivalence relation on P defined by setting $r_1 \approx r_2$ if r_1 and r_2 occur on a cycle of positive edges in $\mathsf{GoR}(P)$. Let $\mathsf{GoR}(P)_\approx$ be the graph defined as follows:

- vertices of $\mathsf{GoR}(P)_\approx$ are equivalence classes defined by the equivalence relation $\approx$, where each equivalence class is denoted as $[r]_\approx = \{r' \in P \mid r \approx r'\}$;
- $\mathsf{GoR}(P)_\approx$ contains an edge $[r_1]_\approx \xrightarrow{\pm} [r_2]_\approx$ whenever $r_1' \xrightarrow{\pm} r_2'$ for some $r_1 \in [r_1]_\approx$ and some $r_2 \in [r_2]_\approx$;
- $\mathsf{GoR}(P)_\approx$ contains an edge $[r_1]_\approx \xrightarrow{-} [r_2]_\approx$ whenever $r_1' \xrightarrow{-} r_2'$ for some $r_1 \in [r_1]_\approx$ and some $r_2 \in [r_2]_\approx$.

Since any cycles in $\mathsf{GoR}(P)$ must consist of positive edges only, $\mathsf{GoR}(P)_\approx$ is a directed acyclic graph. Therefore, there exists an antisymmetric, transitive and total order of the vertices of $\mathsf{GoR}(P)_\approx$, such that for every (positive or negative) edge from vertex v_1 to vertex v_2, v_1 comes before v_2 in the order. It is clear that such an order satisfies the properties of an R-stratification.

Conversely, assume that $\mathsf{GoR}(P)$ has a cycle $r_0 \xrightarrow{\pm} \ldots \xrightarrow{\pm} r_k \xrightarrow{-} r_{k+1} \xrightarrow{\pm} \ldots \xrightarrow{\pm} r_{\ell-1} \xrightarrow{\pm} r_0$. Suppose for a contradiction that P has a stratification $P_1, \ldots, P_n$. For each rule r_i $(i = 0, \ldots, \ell-1)$, let $p(i)$ denote an integer such that $r_i \in P_{p(i)}$. By the properties of R-stratification, we have $p(k) < p(k+1)$ but also $p(i) \leq p((i+1) \bmod \ell)$ for each $i \in \{0, \ldots, \ell-1\}$. The latter implies $p(k+1) \leq p(k)$, which is a contradiction. Hence, P has no R-stratification. □

From the previous result it is clear that, given the graph of reliances, one can decide R-stratification in polynomial time. The overall complexity is therefore dominated by the complexity of checking individual reliances—in this sense, it is polynomial in the total number of rules and coNP-complete only in the maximal size of

a rule. Moreover, in contrast to the NP-completeness of checking positive reliances (Theorem 41), negative reliances can be detected in polynomial time.

Theorem 66. *Given rules r_1 and r_2, it can be decided in polynomial time whether $r_1 \dashrightarrow r_2$. Checking whether a program P is R-stratified is* coNP*-complete.*

Proof. We first show that $r_1 \dashrightarrow r_2$ can be decided in polynomial time. Let r_1 and r_2 be two rules with $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$; w.l.o.g. we assume that $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$. To check $r_1 \dashrightarrow r_2$, we apply the following algorithm: for each $\alpha \in H_1$ and each $\beta \in B_2^-$, check if the following conditions are satisfied:

(i) there exists a most general unifier σ of α and β;

(ii) there are no skolem symbols in $B_1^+\sigma \cup B_2^+\sigma$;

(iii) $(B_1^-\sigma \cup B_2^-\sigma) \cap (B_1^+\sigma \cup B_2^+\sigma) = \emptyset$.

If these conditions hold for at least one pair of α and β, return $r_1 \dashrightarrow r_2$; else return $r_1 \not\dashrightarrow r_2$.

The above algorithm clearly runs in polynomial time: at most $|H_1| \times |B_2^-|$ pairs of atoms need to be considered, their most general unifier can be computed in linear time and the remaining checks are easy to perform in polynomial time.

We now show that the algorithm is correct. For soundness, assume that the algorithm returns $r_1 \dashrightarrow r_2$ and let σ be the unifier considered in the algorithm when terminating. We define a substitution $\theta = \theta_c \circ \sigma$, where θ_c is the substitution that maps every variable x to a fresh constant c_x unique for x and a set of facts $F = B_1^+\theta \cup B_2^+\theta$. It is easy to see that conditions (N1)–(N5) hold for this choice of F and θ.

To show completeness of the algorithm, assume that $r_1 \dashrightarrow r_2$. Then there exist F and θ satisfying the conditions of Definition 59. We can assume that $F = B_1^+\theta \cup B_2^+\theta$, which is w.l.o.g. since only conditions (N1) and (N3) require F to contain atoms. By (N4), there exists $\alpha \in H_1$ and $\beta \in B_2^-$, such that $\alpha\theta = \beta\theta$. Thus, in particular α and β have a most general unifier σ, that is, $\theta = \theta' \circ \sigma$ for some θ'. Thus, since

there are no skolem symbols in $F = B_1^+\theta \cup B_2^+\theta$ by Definition 59, condition (ii) is also satisfied. Likewise, condition (iii) follows from (N2) and (N5).

It remains to show that deciding if a program is R-stratified is coNP-complete. The proof is similar to the proof of coNP-completeness for R-acyclicity in Theorem 41. Membership is immediate by verifying the existence of a problematic cycle as in Proposition 65, which can be done in NP. Hardness was shown in Theorem 41 by constructing two rules of the form $r_1 : \mathsf{Start} \rightarrow H_1$ and $r_2 : B_2 \rightarrow \mathsf{Goal}$ that satisfy the equivalence (4.10). Clearly, (4.10) still holds if we modify r_1 to the rule $r_1' : \mathsf{Start} \wedge \ \mathbf{not}\ \mathsf{Goal} \rightarrow H_1$. Then $r_2 \xrightarrow{-} r_1'$ and hence $\{r_1, r_2\}$ is R-stratified if and only if $r_1 \not\xrightarrow{+} r_2$ which in turn holds if and only if there is no homomorphism from Q to Q'. □

5.2 Computing Stable Models for R-Stratified Programs

It remains to show that R-stratified programs have at most one stable model and that this model can always be obtained by repeated application of rules according to their stratification. This leads to a semi-decision procedure for entailment. If the program is also R-acyclic, we obtain a decision procedure and tight complexity bounds. Please note that R-stratification is a sufficient but not necessary condition for stable model uniqueness, that is there are programs that have a unique stable model but are not R-stratified.

Observe that Definition 59 does not include a condition that corresponds to (P6) from Definition 39. Indeed, as the next example shows, such a condition would not lead to a notion of R-stratification that ensures unique stable models.

Example 67. Given the rules $r_1 :\ \mathbf{not}\ \mathsf{p} \rightarrow \mathsf{q}$ and $r_2 : \mathsf{q} \rightarrow \mathsf{p}$, we find that $r_1 \xrightarrow{+} r_2$ and $r_2 \xrightarrow{-} r_1$, so that the program is not R-stratified. Indeed, it has no stable models for the empty set of facts. Yet, if we would require that $H_2\theta \not\subseteq F$ in Definition 59 then $r_2 \xrightarrow{-} r_1$ would not hold and the program would be R-stratified. Intuitively speaking, negative reliances do not just consider the case where r_2 could derive something new, but also the case where r_2 has already been used in a derivation that is no longer

justified after applying r_1. ♢

We now define a computation scheme that can be used to obtain the unique stable model of R-stratified programs or to derive a contradiction $\bot$ if no such model exists.

Definition 68. *For a set of facts F and a set of rules P with R-stratification $\vec{\mathsf{P}} = P_1, \ldots, P_n$, define $S^0_{\vec{\mathsf{P}}}(F) = F$ and*

$$S^{i+1}_{\vec{\mathsf{P}}}(F) = T^{\infty}_{P_{i+1}}(S^i_{\vec{\mathsf{P}}}(F)) \quad \textit{for} \quad 0 \leq i < n.$$

For the remainder of this section, let P denote an R-stratified program with R-stratification $\vec{\mathsf{P}} = P_1, \ldots, P_n$ and let F denote a set of facts. We use the abbreviations $P^m_1 = \bigcup_{i=1}^m P_i$, $P^0_1 = \emptyset$ and $S^i_{\vec{\mathsf{P}}} = S^i_{\vec{\mathsf{P}}}(F)$. Moreover, for sets of facts F, B^+ and B^-, we define $F \models B^+$ if $B^+ \subseteq F$ and $F \models$ **not** B^- if $B^- \cap F = \emptyset$; we also define $F \models B^+,$ **not** B^- if $F \models B^+$ and $F \models$ **not** B^-.

We first show that $S^n_{\vec{\mathsf{P}}}$ is a (not necessarily unique) stable model of $F \cup P$, provided that $\bot \notin S^n_{\vec{\mathsf{P}}}$. The next two lemmas are key ingredients to this proof. Intuitively speaking, Lemma 69 asserts that, if the body of a rule $r \in P_i$ is satisfied at some point while computing $S^i_{\vec{\mathsf{P}}}$, then it will remain satisfied in all later stages of the computation. The crucial claim is that the negative part of the rule will not be derived at any later stage. The proof of Lemma 69 relies on the definition of $\xrightarrow{-}$.

Lemma 69. *Consider numbers $1 \leq i \leq j \leq k \leq n$ and $\ell \geq 0$, a rule $r \in P_i$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a substitution θ. Then $T^{\ell}_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \models B^+\theta,$ **not** $B^-\theta$ implies $S^k_{\vec{\mathsf{P}}} \models B^+\theta,$ **not** $B^-\theta$.*

Proof. For brevity, define $\mathcal{M} = T^{\ell}_{P_j}(S^{j-1}_{\vec{\mathsf{P}}})$ and $\mathcal{M}' = S^k_{\vec{\mathsf{P}}}$. Let $\mathcal{M} \models B^+\theta,$ **not** $B^-\theta$. Suppose for a contradiction that $\mathcal{M}' \not\models B^+\theta,$ **not** $B^-\theta$. Since $\mathcal{M} \subseteq \mathcal{M}'$, we find that $\mathcal{M}' \models B^+\theta$. Hence $\mathcal{M}' \not\models$ **not** $B^-\theta$, that is, $\mathcal{M}' \cap B^-\theta \neq \emptyset$. Thus there are $m \geq j$ and $o \geq 0$ such that $T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \cap B^-\theta = \emptyset$ and $T^{o+1}_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \cap B^-\theta \neq \emptyset$. Hence there is a rule $r_1 \in P_m$ with $\mathsf{sk}(r_1) = (B^+_1, B^-_1, H_1)$ and a substitution θ_1 such that

$$B^+_1\theta_1 \subseteq T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \tag{5.1}$$

$$B^-_1\theta_1 \cap T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) = \emptyset \tag{5.2}$$

$$H_1\theta_1 \cap B^-\theta \neq \emptyset \tag{5.3}$$

Let $F' = T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}})$; we show that $G = \gamma_{F'}(T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}))$ and $\sigma = \gamma_{F'}(\theta_1 \cup \theta)$ establish the conditions for $r_1 \xrightarrow{-} r$ in Definition 59:

(N1) $B^+_1\sigma \subseteq G$ by (5.1).

(N2) $B^-_1\sigma \cap G = \emptyset$ by (5.2).

(N3) $B^+\sigma \subseteq G$ by $T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \models B^+\theta$ and $T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \subseteq T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}})$; if $m = j$, then $\ell \leq o$ which follows from $T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \cap B^-\theta = \emptyset$ and $T^{o+1}_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \cap B^-\theta \neq \emptyset$.

(N4) $B^-\sigma \cap H_1\sigma \neq \emptyset$ by (5.3); note that for every skolem term $\mathsf{f}(\vec{\mathsf{x}})\theta_1$ in $H_1\theta_1$, the definition of $\gamma_{F'}$ ensures $\gamma_{F'}(\mathsf{f}(\vec{\mathsf{x}})\theta_1) = \mathsf{f}(\gamma_{F'}(\vec{\mathsf{x}}\theta_1))$ (the former occurs in $B^-\sigma$, the latter occurs in $H_1\sigma$).

(N5) $B^-\sigma \cap G = \emptyset$ by $T^o_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \cap B^-\theta = \emptyset$.

Thus $r_1 \xrightarrow{-} r$. However, $r \in P_i$, $r_1 \in P_m$ and $i \leq j \leq m$, which contradicts Definition 61. □

Lemma 70 complements the previous result. Intuitively speaking, it states that a rule $r \in P_i$, which is clearly satisfied after computing $S^i_{\vec{\mathsf{P}}}$, will remain satisfied in all later stages of the computation. The key part of this claim concerns the case that r is satisfied because its positive body is not satisfied. In this case, the positive body will never become satisfied later on, unless the head of the rule becomes satisfied as well. This argument hinges upon the definition of $\xrightarrow{+}$.

Lemma 70. *Consider numbers $1 \leq i < j \leq k \leq n$, a rule $r \in P_i$ and a substitution θ. Then $S^j_{\vec{\mathsf{P}}} \models \mathsf{sk}(r)\theta$ implies $S^k_{\vec{\mathsf{P}}} \models \mathsf{sk}(r)\theta$.*

Proof. Let $\mathsf{sk}(r) = (B^+, B^-, H)$ and suppose for a contradiction that $S^j_{\vec{\mathsf{P}}} \models \mathsf{sk}(r)\theta$ and $S^k_{\vec{\mathsf{P}}} \not\models \mathsf{sk}(r)\theta$. Since $S^j_{\vec{\mathsf{P}}} \subseteq S^k_{\vec{\mathsf{P}}}$, neither $B^-\theta \cap S^j_{\vec{\mathsf{P}}} \neq \emptyset$ (that would imply $B^-\theta \cap S^k_{\vec{\mathsf{P}}} \neq \emptyset$) nor $S^j_{\vec{\mathsf{P}}} \models (B^+ \cup H)\theta$, **not** $B^-\theta$ (that would imply $S^k_{\vec{\mathsf{P}}} \models (B^+ \cup H)\theta$, **not** $B^-\theta$) may hold. Therefore, $B^+\theta \not\subseteq S^j_{\vec{\mathsf{P}}}$. By $S^k_{\vec{\mathsf{P}}} \not\models \mathsf{sk}(r)\theta$, we have $S^k_{\vec{\mathsf{P}}} \models B^+\theta$, **not** $B^-\theta$ and $S^k_{\vec{\mathsf{P}}} \not\models H\theta$. As a consequence, there exists a maximal non-empty set of facts A such that $A \subseteq B^+\theta$ and $A \cap S^j_{\vec{\mathsf{P}}} = \emptyset$. Hence there are numbers $\ell \geq 0$ and m such that

$j < m \leq k$, a rule $r_1 \in P_m$ with $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and a substitution θ_1 such that

$$B_1^+\theta_1 \subseteq T_{P_m}^{\ell}(S_{\vec{\mathsf{P}}}^{m-1}) \tag{5.4}$$

$$B_1^-\theta_1 \cap T_{P_m}^{\ell}(S_{\vec{\mathsf{P}}}^{m-1}) = \emptyset \tag{5.5}$$

$$H_1\theta_1 \cap A \not\subseteq T_{P_m}^{\ell}(S_{\vec{\mathsf{P}}}^{m-1}) \tag{5.6}$$

Let $F' = T_{P_m}^{\ell}(S_{\vec{\mathsf{P}}}^{m-1}) \cup (A \setminus H_1\theta_1)$. We claim that the set of facts $G = \gamma_{F'}(F')$ and the substitution $\sigma = \gamma_{F'}(\theta_1 \cup \theta)$ meet the conditions for $r_1 \xrightarrow{+} r$ in Definition 39.

(P1) $B_1^+\sigma \subseteq G$ by (5.4).

(P2) $B_1^-\sigma \cap G = \emptyset$ by (5.5) and $B_1^-\theta_1 \cap A = \emptyset$; note that the latter follows from $A \subseteq S_{\vec{\mathsf{P}}}^k$ and $S_{\vec{\mathsf{P}}}^k \models B_1^+\theta_1$, **not** $B_1^-\theta_1$ which is a consequence of Lemma 69, $T_{P_m}^{\ell}(S_{\vec{\mathsf{P}}}^{m-1}) \models B_1^+\theta_1$, **not** $B_1^-\theta_1$, $r_1 \in P_m$ and $k \geq m$.

(P3) $B^+\sigma \subseteq G \cup H_1\sigma$ by $B^+\theta \subseteq F' \cup H_1\theta_1$.

(P4) $B^-\sigma \cap (G \cup H_1\sigma) = \emptyset$ by $S_{\vec{\mathsf{P}}}^k \models B^+\theta$, **not** $B^-\theta$.

(P5) $B^+\sigma \not\subseteq G$ by (5.6).

(P6) $H\sigma \not\subseteq G \cup H_1\sigma$ by $G \cup H_1\theta_1 \subseteq S_{\vec{\mathsf{P}}}^k$ and $S_{\vec{\mathsf{P}}}^k \not\models H\theta$.

Thus, by $r_1 \in P_m$, $r \in P_i$ and $m > i$ we derive a contradiction and show our initial claim. □

Using Lemmas 69 and 70, we can show the following result.

Proposition 71. *If $\bot \notin S_{\vec{\mathsf{P}}}^n$, then $S_{\vec{\mathsf{P}}}^n \models_{\mathsf{SM}} F \cup P$.*

Proof. We show that $\bot \notin S_{\vec{\mathsf{P}}}^n$ implies (♣) $S_{\vec{\mathsf{P}}}^n$ is a model of $F \cup \mathsf{GL}(P_1^n, S_{\vec{\mathsf{P}}}^n)$ and (♠) the model $S_{\vec{\mathsf{P}}}^n$ of $F \cup \mathsf{GL}(P_1^n, S_{\vec{\mathsf{P}}}^n)$ is minimal.

(♣) We prove $S_{\vec{\mathsf{P}}}^k \models F \cup \mathsf{GL}(P_1^k, S_{\vec{\mathsf{P}}}^k)$ for all $k \in \{0, \ldots, n\}$ by induction over k.

If $k = 0$, then $P_1^k = \emptyset$ and $S_{\vec{\mathsf{P}}}^k = F$. Clearly, F is a stable model of F, since $\mathsf{GL}(\emptyset, F) = \emptyset$ and $T_{\emptyset}^{\infty}(F) = F$.

For the induction step, let $S^k_{\vec{\mathsf{P}}}$ be a model of $F \cup \mathsf{GL}(P_1, S^k_{\vec{\mathsf{P}}})$ and we prove that $S^{k+1}_{\vec{\mathsf{P}}}$ is a model of $F \cup \mathsf{GL}(P_1^{k+1}, S^{k+1}_{\vec{\mathsf{P}}})$. Note that $S^{k+1}_{\vec{\mathsf{P}}} \models F \cup \mathsf{GL}(P_1^{k+1}, S^{k+1}_{\vec{\mathsf{P}}})$ is equivalent to $S^{k+1}_{\vec{\mathsf{P}}} \models F \cup \mathsf{sk}(P_1^{k+1})$. Since $S^{k+1}_{\vec{\mathsf{P}}} = T^\infty_{P_{k+1}}(S^k_{\vec{\mathsf{P}}})$, we find $S^{k+1}_{\vec{\mathsf{P}}} \models F \cup \mathsf{sk}(P_{k+1})$. It remains to show $S^{k+1}_{\vec{\mathsf{P}}} \models \mathsf{sk}(P_1^k)$. Thus consider an arbitrary rule $r \in P_1^k$. By induction hypothesis, $S^k_{\vec{\mathsf{P}}} \models \mathsf{sk}(P_1^k)$ and thus $S^k_{\vec{\mathsf{P}}} \models \mathsf{sk}(r)$. By Lemma 70, $S^{k+1}_{\vec{\mathsf{P}}} \models \mathsf{sk}(r)$. Since r was arbitrary, this shows the claim.

(♠) Suppose for a contradiction that there is set of facts $\mathcal{M} \subsetneq S^n_{\vec{\mathsf{P}}}$ such that $\mathcal{M} \models F \cup \mathsf{GL}(P_1^n, S^n_{\vec{\mathsf{P}}})$. Then there are $\ell \geq 0$ and j, where $1 \leq j \leq n$, such that $T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \subseteq \mathcal{M}$ and $T^{\ell+1}_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \not\subseteq \mathcal{M}$. Thus there is a rule $r \in P_j$ with $\mathsf{sk}(r) = (B^+, B^-, H)$, and a substitution θ with

$$B^+\theta \subseteq T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \tag{5.7}$$

$$B^-\theta \cap T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) = \emptyset \tag{5.8}$$

$$H\theta \not\subseteq \mathcal{M} \tag{5.9}$$

By $T^\ell_{P_j}(S^{j-1}_{\vec{\mathsf{P}}}) \subseteq \mathcal{M}$ and (5.7), we infer that $B^+\theta \subseteq \mathcal{M}$. Together with (5.9) and $\mathcal{M} \models F \cup \mathsf{GL}(P_1^n, S^n_{\vec{\mathsf{P}}})$, this implies $(B^+\theta, \emptyset, H\theta) \notin \mathsf{GL}(P_1^n, S^n_{\vec{\mathsf{P}}})$ and so $B^-\theta \cap S^n_{\vec{\mathsf{P}}} \neq \emptyset$. However, by (5.7), (5.8) and Lemma 69, $S^n_{\vec{\mathsf{P}}} \models B^+\theta$, **not** $B^-\theta$, which yields the required contradiction. □

The next lemma captures the main statement that is needed to prove the claimed uniqueness of the stable model. The statement is that every stable model of $P \cup F$ coincides with $S^n_{\vec{\mathsf{P}}}$ and the implication is demonstrated by showing inductively that, for all $k \in \{0, \ldots, n\}$, $S^k_{\vec{\mathsf{P}}} = T^\infty_{\mathsf{GL}(P_1^k, \mathcal{M})}(F)$.

Lemma 72. *If $\mathcal{M} \models_{\mathsf{SM}} P \cup F$, then $S^n_{\vec{\mathsf{P}}} = \mathcal{M}$.*

Proof. We show by induction that, for each $k \in \{0, \ldots, n\}$, it is $S^k_{\vec{\mathsf{P}}} = T^\infty_{\mathsf{GL}(P_1^k, \mathcal{M})}(F)$. This establishes the claim since the latter is equal to $\mathcal{M}$ if $k = n$.

For $k = 0$, the claim is immediate: we have $S^0_{\vec{\mathsf{P}}} = F$, $P_1^0 = \emptyset$ and $\mathsf{GL}(P_1^0, \mathcal{M}) = \emptyset$ that imply $T^\infty_{\mathsf{GL}(P_1^0, \mathcal{M})}(F) = F$.

For the induction step, assume that the claim has been shown for k. We show that $S^{k+1}_{\vec{\mathsf{P}}} = T^\infty_{\mathsf{GL}(P_1^{k+1}, \mathcal{M})}(F)$.

We first show that $S^{k+1}_{\vec{\mathsf{P}}} \subseteq T^{\infty}_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$. For brevity, we use T^i to denote $T^{\infty}_{\mathsf{GL}(P_1^i,\mathcal{M})}(F)$. Suppose for a contradiction that $S^{k+1}_{\vec{\mathsf{P}}} \not\subseteq T^{k+1}$. Then there are numbers $m \in \{1,\ldots,k+1\}$ and $\ell \geq 0$ such that

$$T^{\ell}_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \subseteq T^{k+1} \tag{5.10}$$

$$T^{\ell+1}_{P_m}(S^{m-1}_{\vec{\mathsf{P}}}) \not\subseteq T^{k+1} \tag{5.11}$$

By the induction hypothesis, $S^{k}_{\vec{\mathsf{P}}} \subseteq T^{k+1}$, so (5.11) implies $m = k+1$. Thus there is a rule $r \in P_{k+1}$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a substitution θ such that

$$B^+\theta \subseteq T^{\ell}_{P_{k+1}}(S^{k}_{\vec{\mathsf{P}}}) \tag{5.12}$$

$$B^-\theta \cap T^{\ell}_{P_{k+1}}(S^{k}_{\vec{\mathsf{P}}}) = \emptyset \tag{5.13}$$

$$H\theta \not\subseteq T^{k+1} \tag{5.14}$$

Together, (5.12), (5.10) (where $m = k+1$) and (5.14) imply

$$B^-\theta \cap T^{k+1} \neq \emptyset \tag{5.15}$$

Thus there is a rule r_1 with $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$, a substitution θ_1 and a number j such that

$$B_1^+\theta_1 \subseteq T^{j}_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \tag{5.16}$$

$$B_1^-\theta_1 \cap \mathcal{M} = \emptyset \tag{5.17}$$

$$B^-\theta \cap T^{j}_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) = \emptyset \tag{5.18}$$

$$B^-\theta \cap H_1\theta_1 \neq \emptyset \tag{5.19}$$

Define $F' = T^{j}_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \cup T^{\ell}_{P_{k+1}}(S^{k}_{\vec{\mathsf{P}}})$. We claim that the set of facts $G = \gamma_{F'}(F')$ and substitution $\sigma = \gamma_{F'}(\theta \cup \theta_1)$ satisfy the conditions for $r_1 \xrightarrow{-} r$ in Definition 59. By definition, G does not contain function symbols.

(N1) $B_1^+\sigma \subseteq G$ by (5.16).

(N2) $B_1^-\sigma \cap G = \emptyset$ by (5.17) and $F' \subseteq \mathcal{M}$, obtained from (5.10) (where $m = k+1$).

(N3) $B^+\sigma \subseteq G$ by (5.12).

(N4) $B^-\sigma \cap H_1\sigma \neq \emptyset$ by (5.19).

(N5) $B^-\sigma \cap G = \emptyset$ by (5.13) and (5.18).

Thus $r_1 \xrightarrow{-} r$ and therefore $r_1 \in P_1^k$ by Definition 61. By (5.17) and $T^k \subseteq \mathcal{M}$, we have $B_1^-\theta_1 \cap T^k = \emptyset$. By the induction hypothesis, $T^k = S_{\vec{\mathsf{P}}}^k$, so $B_1^-\theta_1 \cap S_{\vec{\mathsf{P}}}^k = \emptyset$. By (5.19) and (5.13), $H_1\theta_1 \not\subseteq S_{\vec{\mathsf{P}}}^k$. Together, these observations imply that $B_1^+\theta_1 \not\subseteq S_{\vec{\mathsf{P}}}^k$. Using again $T^k = S_{\vec{\mathsf{P}}}^k$, we find that $B_1^+\theta_1 \not\subseteq T^k$.

From $B_1^+\theta_1 \not\subseteq T^k$ and (5.16), we conclude that there is a rule $r_2 \in P_{k+1}$ with $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$, a substitution θ_2 and a number $o < j$ such that

$$B_2^+\theta_2 \subseteq T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \tag{5.20}$$

$$B_2^-\theta_2 \cap \mathcal{M} = \emptyset \tag{5.21}$$

$$H_2\theta_2 \cap B_1^+\theta_1 \not\subseteq T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \tag{5.22}$$

By (5.18) and (5.19), there is $\alpha \in H_1\theta_1 \setminus T^j_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$, and by (5.22), there is $\beta \in (H_2\theta_2 \cap B_1^+\theta_1) \setminus T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$. Let $F'' = \mathcal{M} \setminus \{\alpha, \beta\}$. We claim that the set of facts $G' = \gamma_{F''}(F'')$ and substitution $\sigma' = \gamma_{F''}(\theta_1 \cup \theta_2)$ satisfy the conditions for $r_2 \xrightarrow{+} r_1$ in Definition 39. By definition, G' does not contain function symbols.

(P1) $B_2^+\sigma' \subseteq G'$ by (5.20) and $T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \subseteq F''$. For the latter, note that $\beta \notin T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$ and that $\alpha \notin T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$ since $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$ and $T^o_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F) \subseteq T^j_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$ (by $o < j$).

(P2) $B_2^-\sigma' \cap G' = \emptyset$ by (5.21).

(P3) $B_1^+\sigma' \subseteq G' \cup H_2\sigma'$ by (5.16) and $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$.

(P4) $B_1^-\sigma' \cap (G' \cup H_2\sigma') = \emptyset$ by (5.17) and $H_2\theta_2 \subseteq \mathcal{M}$.

(P5) $B_1^+\sigma' \not\subseteq G'$ by our choice of β.

(P6) $H_1\sigma' \not\subseteq G' \cup H_2\sigma'$ by our choice of α and $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1},\mathcal{M})}(F)$.

Thus $r_2 \xrightarrow{+} r_1$. This contradicts the assumption that $r_1 \in P_1^k$ while $r_2 \in P_{k+1}$, and thus refutes our initial assumption that $S_{\vec{\mathsf{P}}}^{k+1} \not\subseteq T^{k+1}$. We have thus shown that $S_{\vec{\mathsf{P}}}^{k+1} \subseteq T^{k+1}$.

For the converse, recall that $S^{k+1}_{\vec{\mathsf{P}}}$ is a stable model of P_1^{k+1} (by Proposition 71 and $\bot \notin S^{k+1}_{\vec{\mathsf{P}}}$ which is a consequence of $\bot \notin T^{k+1}$). Thus $S^{k+1}_{\vec{\mathsf{P}}} = T^{\infty}_{\mathsf{GL}(P_1^{k+1}, S^{k+1}_{\vec{\mathsf{P}}})}(F)$. Now since $S^{k+1}_{\vec{\mathsf{P}}} \subseteq T^{k+1} \subseteq \mathcal{M}$, we find that

$$\mathsf{GL}(P_1^{k+1}, S^{k+1}_{\vec{\mathsf{P}}}) \supseteq \mathsf{GL}(P_1^{k+1}, T^{k+1}) \supseteq \mathsf{GL}(P_1^{k+1}, \mathcal{M})$$

Thus $S^{k+1}_{\vec{\mathsf{P}}} = T^{\infty}_{\mathsf{GL}(P_1^{k+1}, S^{k+1}_{\vec{\mathsf{P}}})}(F) \supseteq T^{\infty}_{\mathsf{GL}(P_1^{k+1}, \mathcal{M})}(F) = T^{k+1}$ as required. □

Summing up, we obtain the main result of this section, according to which stable models of R-stratified programs are unique.

Theorem 73. *If $\bot \notin S^n_{\vec{\mathsf{P}}}$, then $S^n_{\vec{\mathsf{P}}}$ is the unique stable model of $F \cup P$. Otherwise $F \cup P$ does not have a stable model.*

Proof. Assume that $\bot \notin S^n_{\vec{\mathsf{P}}}$. By Proposition 71, $S^n_{\vec{\mathsf{P}}} \models_{\mathsf{SM}} F \cup P$. By Lemma 72, every stable model of $P \cup F$ is equal to $S^n_{\vec{\mathsf{P}}}$, which establishes uniqueness.

If $\bot \in S^n_{\vec{\mathsf{P}}}$, then let $P' = P \setminus P_C$, where P_C is the set of constraints that are in P. P' is clearly R-stratified, so $S^n_{\vec{\mathsf{P}}} \setminus \bot$ is the unique stable model of $F \cup P'$. Since there exists at least one constraint $c \in P_C$ such that $S^n_{\vec{\mathsf{P}}} \not\models c$, $F \cup P$ does not have a stable model. □

5.3 Complexity of Reasoning for R-Acyclic, R-Stratified Programs

We can further improve the complexity results of Theorem 45 for programs that are both R-acyclic and R-stratified. The Turing machine reduction used to show Theorem 45 can directly be used to show hardness: the constructed program is R-stratified precisely if the Turing machine is deterministic.

Theorem 74. *Let P be an R-acyclic, R-stratified program, let F be a set of facts and let α be a fact. Deciding $P \cup F \models \alpha$ is* 2ExpTime*-complete w.r.t. program complexity and* P*-complete w.r.t. data complexity.*

Proof. In the proof of Theorem 45, we have already shown that there is a maximal number n of terms that may occur in any stable model of an R-acyclic program P,

where n is double exponential in the size of $P \cup F$ and polynomial in the size of F. If a is the maximal arity of predicates in P and b is the number of predicate symbols, then the size of the computed stable model is bounded by $b \cdot n^a$, which is still double exponential in $P \cup F$ and polynomial in F. By Theorem 73, we can compute the unique stable model of $P \cup F$ as in Definition 68, where we only need to consider substitutions that map to the bounded set of relevant ground terms. The applicability of one ground rule over a set of facts of size at most $b \cdot n^a$ can be checked deterministically in time $b \cdot n^a$. As discussed in the proof of Theorem 45, the number of ground rules considered in each step is again double exponential in the size of $P \cup F$ and polynomial in the size of F; the number of steps has the same bound, since each rule can be applied at most once. Hence, $S^n_{\vec{\mathsf{P}}}$ can be computed in time that is double exponential in the size of $P \cup F$ and polynomial in the size of F and the claimed upper bounds for complexity follow.

Hardness for 2ExpTime w.r.t. the size of $P \cup F$ follows by the same Turing machine construction as in the proof of Theorem 45, applied to deterministic Turing machines only. The resulting program is R-stratified since the transition rules do not contain negation for deterministic Turing machines. Hardness for P w.r.t. the size of F follows from P-hardness w.r.t. data complexity of fact entailment for datalog rules (i.e., rules without negation, existential quantifiers or function symbols) [65]. □

5.4 Revisiting Stratification Conditions

From a logic programming point of view, stratification (as specified in Definition 12, also known and as *predicate* stratification) has been acknowledged as a rather restrictive condition for identifying logic programs with unique stable models [9]. As a consequence, several criteria that strictly extend stratification were proposed in the nineties, such as *local stratification* [200], *weak stratification* [199], *effective stratification* [26], *modular stratification* [210] and *left-to-right dynamic stratification* [213].

Local stratification generalises stratification by considering the (infinite) groundings of normal logic programs [200, 92]. This condition is defined for programs that

may contain function symbols either in the body or in the head and, thus, are different from the programs considered here where function symbols only occur in the head as a result of skolemisation. Additionally, Przymusinski defines the *perfect model semantics*—a semantics that characterises locally stratified programs.

Definition 75 (Local Stratification)**.** *Given a program P, a sequence of disjoint programs $\vec{\mathsf{P}} = P_1, \ldots, P_n$ is a* local stratification *of P if $\mathsf{Ground}(P) = \bigcup_{i=1}^{n} P_i$ and, for all programs $P_i, P_j \in \vec{\mathsf{P}}$, rules $(B_1^+, B_1^-, H_1) \in P_i$ and $(B_2^+, B_2^-, H_2) \in P_j$ and every atom $\alpha \in \mathsf{Pred}(H_1)$, we have: (i) if $\alpha \in \mathsf{Pred}(B_2^+)$ then $i \leq j$ and (ii) if $\alpha \in \mathsf{Pred}(B_2^-)$ then $i < j$. P is* locally stratified *if it has a local stratification.*

Local stratification is undecidable [59] and does not generalise R-stratification as Example 76 below shows. Moreover, Example 76 demonstrates that R-stratification strictly extends predicate stratification, even when considering the restricted case of datalog programs with nonmonotonic negation.

Example 76. Let r_1 and r_2 be defined as follows:

$$r_1 : \quad \mathsf{A}(\mathsf{x}) \wedge \mathsf{S}(\mathsf{x},\mathsf{y}) \wedge \ \mathbf{not}\ \mathsf{A}(\mathsf{y}) \wedge \ \mathbf{not}\ \mathsf{B}(\mathsf{y}) \rightarrow \mathsf{R}(\mathsf{x},\mathsf{y})$$

$$r_2 : \quad \mathsf{A}(\mathsf{u}) \wedge \mathsf{T}(\mathsf{u},\mathsf{v}) \wedge \ \mathbf{not}\ \mathsf{R}(\mathsf{v},\mathsf{u}) \rightarrow \mathsf{B}(\mathsf{u})$$

We have $r_1 \not\rightarrow r_2$ because there are no F and θ that satisfy (N1)–(N5) for $r_1 \dashrightarrow r_2$, as this requires $\theta(\mathsf{u}) = \theta(\mathsf{y})$, $\mathsf{A}(\mathsf{u})\theta \in F$ but $\mathsf{A}(\mathsf{y})\theta \notin F$; by $\mathsf{Pred}(H_1) \cap \mathsf{Pred}(B_2^+) = \emptyset$ it is $r_1 \overset{+}{\not\rightarrow} r_2$. Moreover, by $r_1 \not\rightarrow r_1$ and $r_2 \not\rightarrow r_2$, we derive that $P = \{r_1, r_2\}$ is R-stratified. P is not locally stratified as shown by the following instantiation of P.

$$r_1' : \quad \mathsf{A}(\mathsf{a}) \wedge \mathsf{S}(\mathsf{a},\mathsf{b}) \wedge \ \mathbf{not}\ \mathsf{A}(\mathsf{b}) \wedge \ \mathbf{not}\ \mathsf{B}(\mathsf{b}) \rightarrow \mathsf{R}(\mathsf{a},\mathsf{b})$$

$$r_2' : \quad \mathsf{A}(\mathsf{b}) \wedge \mathsf{T}(\mathsf{b},\mathsf{a}) \wedge \ \mathbf{not}\ \mathsf{R}(\mathsf{a},\mathsf{b}) \rightarrow \mathsf{B}(\mathsf{b})$$

Since $\mathsf{B}(\mathsf{b})$ occurs both in the negative body of r_1' and in the head of r_2' and $\mathsf{R}(\mathsf{a},\mathsf{b})$ occurs both in the negative body of r_2' and in the head of r_1', it is not possible to define a local stratification for $\{r_1', r_2'\}$. ♢

Further to that, Przymusinska and Przymusinski define the class of *weakly stratified* logic programs, which properly contains the class of locally stratified logic programs [199, 200]. Additionally, a *weakly perfect model* semantics is described, such

that if a program is weakly stratified, then it has a unique weakly perfect model; however, the class of programs with a weakly perfect model is strictly broader than the class of programs that are weakly stratified. Similarly, the class of programs that are weakly stratified is strictly contained in the class of programs with a unique stable model. Weak stratification of a program can be checked by inspecting how an iteration that involves construction of models ends, where the construction takes countably many steps [199]. Finally, weak stratification is studied for programs with arbitrary use of function symbols, but no complexity bounds are provided.

In the same vein, *modularly stratified* programs were introduced as an extension of locally stratified programs [210]. In a nutshell, a program is modularly stratified if after grounding and splitting the program into components and removing those rules whose body cannot be satisfied, one ends up with mutually recursive components that are locally stratified. Ross shows that every modularly stratified program has a unique stable model [210]. However, modular stratification is considered only for function-free programs and without any complexity results for testing it; in fact, Sagonas et al. state that checking modular stratification is undecidable and prove that weakly stratified programs are strictly more general than modularly stratified programs [213].

In addition to the above, *left-to-right dynamically* (LRD) stratified programs [213] extend stratified programs but under the well-founded semantics [90]. The idea of dynamic stratification consists in constructing the components of the program dynamically so that atom dependencies that can never be satisfied are eliminated. Sagonas et al. prove that LRD stratified programs strictly contain the modularly stratified programs; nonetheless, no complexity bounds are established for checking left-to-right dynamic stratification [213].

An even more general notion of stratification that strictly captures weak stratification is suggested by Bidoit and Froideveaux, who define *effectively stratified* programs, a class of programs with a unique stable model [26]. Intuitively, effective stratification is decided in two steps: first, the logic program is instantiated and, then, the 'relevant' ground rules are extracted, where the relevance of the rules depends on the

facts contained in the program. An interesting feature of effective stratification is that it is data dependent, i.e. a program might no longer be effectively stratified after updating its set of facts. Bidoit and Froideveaux consider function-free rules, but do not pinpoint the complexity for checking effective stratification [26].

Our notion of R-stratification can be combined with many existing works to obtain more general criteria. In particular, the graph of reliances can be used to induce a partition of a program into sub-programs so that each problematic reliance cycle is fully contained in some part. One can then apply other acyclicity or stratification criteria to each part to extend our results to even wider program classes. As we have already seen, Deutsch et al. exploit this idea to obtain broader classes of acyclic programs with the help of weak acyclicity [71]. It would be interesting to apply similar techniques to design more comprehensive stratification conditions.

Regarding the suitability of different stratification conditions for different use cases, it is difficult to favour one stratification criterion over another, mostly due to the lack of complexity results for previously suggested generalisations of stratification. This is further exacerbated by the undecidability of some of these conditions in the presence of function symbols. For the purposes of structure-based classification which is the main focus of this work, we adopt R-stratification because it provides us with a deterministic and straightforward to implement reasoning algorithm.

5.5 Related Work on Nonmonotonic Rules

Eiter et al. [77] define *description logic programs* (dl-programs for short), which is a unifying framework for integrating DL knowledge bases and logic programs with negation-as-failure. A suite of semantics is suggested for these programs such as strong and weak answer set semantics and it is shown how dl-programs under these semantics can be used to support closed-world reasoning. Moreover, complexity results are provided for reasoning over dl-programs. For a knowledge base $KB = (L, P)$, where L is a set of DL axioms and P is a logic program, answer set existence is proved to be ExpTime-complete if L is in $\mathcal{SHIF}(D)$ and KB is positive or stratified and

NExpTime-complete if KB is arbitrary; if L is in $\mathcal{SHOIN}(D)$, answer set existence becomes NExpTime-complete if KB is positive and $\mathrm{P}^{\mathrm{NExpTime}}$-complete if KB is stratified and arbitrary. Since dl-programs consist only of function-free rules, they do not contain function symbols and thus are not well-suited for performing the reasoning tasks discussed in Chapter 3.

Lukasiewicz [163] suggests a further approach for rules with nonmonotonic negation that combines disjunctive logic programs with DLs. A new semantics is provided that faithfully extends both disjunctive logic programs under answer set semantics and DLs under the standard first-order semantics. Also, algorithms are designed for answer set existence and brave/cautious reasoning and a computational analysis of the suggested framework is conducted. However, only function-free rules are considered which distinguishes this work from our approach.

In a similar spirit, Eiter et al. [76] introduce well-founded semantics for dl-programs. Under the well-founded semantics, it is proved that literal inference for dl-programs w.r.t. program complexity is ExpTime-complete for $\mathcal{SHIF}(D)$ and $\mathrm{P}^{\mathrm{NExpTime}}$-complete for $\mathcal{SHOIN}(D)$; the same problem w.r.t. data complexity is shown to be Δ_2^P-complete for $\mathcal{SHIF}(D)$ and $\mathcal{SHOIN}(D)$. Moreover, two interesting data tractable cases are singled out: the abovementioned problem becomes (i) P-complete when all dl-queries in a dl-program can be evaluated in polynomial time and (ii) in LogSpace when evaluation of dl-queries in a dl-program is first-order rewritable and the dl-program is acyclic. In the considered dl-programs no function symbols occur, which differentiates the suggested formalism from our framework.

A first step towards enriching rules with existentially quantified variables in the head is taken by Calì et al. [41] who introduce the family of Datalog$^{\pm}$ formalisms. It is shown that fact inference for guarded Datalog$^{\pm}$ rules is P-complete w.r.t. data complexity; it is also proved that boolean conjunctive query answering for Datalog$^{\pm}$ rules with a single body atom is in AC_0 w.r.t. data complexity. Furthermore, these results are preserved when rules are extended with stratified non-monotonic negation. This is of interest to our work, but as discussed in Section 3.3.1, non-guarded rules are essential for the representation of our domain of graph-shaped structures. Moreover,

as it is shown in Section 3.3.2, non-stratified negation is necessary too.

In order to extend Datalog$^{\pm}$ with non-stratified negation, Gottlob et al. [96] define a new well-founded semantics for Datalog$^{\pm}$ that adopts the unique name assumption (UNA) and is called equality-friendly well-founded semantics (EFWFS). It is thus shown that answering a normal boolean conjunctive query (i.e. containing both positive and negative atoms) in guarded Datalog$^{\pm}$ under EFWFS is 2ExpTime-complete in the general case and ExpTime-complete in the case of bounded arities and acyclic queries. However, this approach is restricted to guarded rules and non-guardedness is indispensable for representing objects of our domain.

In the same vein, Hernich et al. [116] define an additional well-founded semantics for non-stratified Datalog$^{\pm}$ rules but this time with the UNA. In this work, it is shown that answering normal boolean conjunctive queries for guarded Datalog$^{\pm}$ under well-founded semantics with UNA is 2ExpTime-complete (ExpTime-complete for predicates of bounded arity) w.r.t. combined complexity and P-complete w.r.t. data complexity. Similarly to before, the formalism is restricted to guarded rules and thus it does not fulfill the requirements for representing structured objects.

Chapter 6

Enhancing Acyclicity and Stratification with Constraints

We next propose a generalisation of R-acyclicity and R-stratification by examining the structure not only of the rules, but also of the facts that the program contains. To this end, we relax the notions of positive and negative reliances using *constraints* that capture all sets of facts that e.g. arise in the context of a given application. Part of the results presented in this chapter are published in [170].

We start this chapter by defining in Section 6.1 positive and negative reliances under constraints and by pinpointing the computational complexity for checking such reliances. Then, in Section 6.2, we introduce R-acyclicity under constraints and, in Section 6.3, R-stratification under constraints; we establish tight complexity results for checking both conditions and provide examples which demonstrate that our newly defined acyclicity and stratification notions properly extend the previous ones. In Section 6.4 we recapitulate and discuss the derived complexity results for decision problems emerging in the context of reliances and discuss some related work.

6.1 Positive and Negative Reliances under Constraints

To widen the classes of recognisable logic programs with unique stable models, it has been proposed to study stratification for a particular set of facts [26]. However,

notions that depend on a particular set of facts do not easily capture a wider class of relevant sets of facts, making it hard to develop logic programs that are robust to changing inputs. In order to overcome this we use *constraints*, that is, rules of the form $B^+ \rightarrow \bot$ where B^+ is a set of atoms. As illustrated by the following example, constraints restrict the possible types of input so that more programs are stratified.

Example 77. Let $P = \{r_1, r_2, r_3\}$ state that organic molecules are those containing carbon and that each inorganic entity is a molecule of geological origin:

$$r_1 : \quad \mathsf{molecule(x)} \wedge \mathsf{hasAtom(x,y)} \wedge \mathsf{c(y)} \rightarrow \mathsf{organic(x)}$$

$$r_2 : \quad \mathsf{molecule(x)} \wedge \ \mathbf{not}\ \mathsf{organic(x)} \rightarrow \mathsf{inorganic(x)}$$

$$r_3 : \quad \mathsf{inorganic(x)} \rightarrow \mathsf{molecule(x)} \wedge \mathsf{geoOrigin(x)}$$

It is easily checked that $r_1 \xrightarrow{-} r_2 \xrightarrow{+} r_3 \xrightarrow{+} r_1$, so P is not R-stratified by Proposition 65. Although the program has a unique stable model for all sets of facts, there is no order of rule applications arising from an R-stratification that produces the stable model in all cases. In particular, the set of facts $\{\mathsf{inorganic(a)}, \mathsf{hasAtom(a,b)}, \mathsf{c(b)}\}$ requires us to apply r_3 before r_1. This situation is undesired, since inorganic molecules usually do not contain carbon and a refined notion of reliance should take this into account. ♢

To formalise this idea, we extend our earlier definitions of positive and negative reliances with constraints.

Definition 78 (Reliances under Constraints)**.** *Let r_1 and r_2 be rules and let C be a set of positive constraints.*

- r_2 positively relies on r_1 under C *(written $r_1 \xrightarrow{+}_C r_2$) if there exists a set of facts F and a substitution θ that satisfy the conditions in Definition 39 and where $F \models C$.*
- r_2 negatively relies on r_1 under C *(written $r_1 \xrightarrow{-}_C r_2$) if there exists a set of facts F and a substitution θ that satisfy the conditions in Definition 59 and where $F \models C$.*

The classes of programs that are R-acyclic under C *and* R-stratified under C *are defined as in Definition 42 and 61, respectively, but using* $\stackrel{+}{\rightarrow}_C$ *instead of* $\stackrel{+}{\rightarrow}$.

It should be noted that our earlier results treat constraints like any other rule of P. When considering reliances under constraints, it is still possible to treat some constraints of the input as part of the program P, especially if these constraints are not deemed to be relevant for showing stratification. Indeed, the fewer constraints are part of C, the fewer additional tests are needed to check reliances.

Example 79. Consider the rules of Example 77 and the constraint

$$c : \mathsf{inorganic}(\mathsf{x}) \wedge \mathsf{hasAtom}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{c}(\mathsf{y}) \rightarrow \bot$$

With $C = \{c\}$, we find $r_3 \stackrel{+}{\not\rightarrow}_C r_1$ and indeed $P_1 = \{r_1\}$, $P_2 = \{r_2, r_3\}$ is an R-stratification of $\{r_1, r_2, r_3\}$ under these constraints. ♢

The consideration of constraints increases the complexity of checking positive reliances from NP to Σ_2^P, i.e. the check can be performed in polynomial time by a nondeterministic Turing machine using an NP oracle. Yet, as before, the NP computations correspond to checking the applicability of a rule or constraint to a small set of facts, for which efficient implementations exist. A lower bound can be shown by reducing satisfiability of a quantified Boolean formula $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ to testing a positive reliance under a set of constraints (our hardness proof is original).

Theorem 80. *Given rules* r_1 *and* r_2 *and a set of constraints* C*, deciding whether* $r_1 \stackrel{+}{\rightarrow}_C r_2$ *is* Σ_2^P*-complete.*

Proof. We first show the complexity for deciding $r_1 \stackrel{+}{\rightarrow}_C r_2$.

(Membership) As in the proof of Theorem 41, we can guess a set of facts F and a substitution θ with size polynomial in the input and then check in polynomial time whether the conditions of Definition 39 are met. Given a constraint $c \in C$, we can check $F \models c$ by invoking a (co)NP oracle that decides the existence of a substitution under which the body of c is included in F (which would show $F \not\models c$). Since the number of oracle calls is linear in $|C|$, the overall check is in Σ_2^P.

(Hardness) We consider quantified Boolean formulae (QBF) of the form $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$, where $\vec{\mathsf{p}}$ and $\vec{\mathsf{q}}$ are lists of propositional variables and φ is a propositional formula over $\vec{\mathsf{p}} \cup \vec{\mathsf{q}}$. Deciding the satisfiability of such a QBF is Σ_2^P-hard, even if it is of the form

$$\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.(\ell_{11} \wedge \ell_{12} \wedge \ell_{13}) \vee \ldots \vee (\ell_{n1} \wedge \ell_{n2} \wedge \ell_{n3})$$

where each ℓ_{ij} is a propositional variable or a negated propositional variable [22]. Given a QBF $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ of this form, we construct rule r_1, rule r_2 and constraints C such that

$$\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi \text{ is satisfiable} \quad \Leftrightarrow \quad r_1 \xrightarrow{+}_C r_2 \tag{6.1}$$

Let $\vec{\mathsf{p}} = \langle p_1, \ldots, p_k\rangle$ and $\vec{\mathsf{q}} = \langle q_1, \ldots, q_m\rangle$. Rules r_1 and r_2 are defined as follows, where 0 and 1 are constants and where all other terms are variables:

r_1:
$$\begin{aligned}
&\mathsf{V}(\mathsf{x}_1, \ldots, \mathsf{x}_k) \wedge \\
&\mathsf{N}(0,0,0) \wedge \mathsf{N}(0,0,1) \wedge \mathsf{N}(0,1,0) \wedge \\
&\mathsf{N}(0,1,1) \wedge \mathsf{N}(1,0,0) \wedge \mathsf{N}(1,0,1) \wedge \mathsf{N}(1,1,0) \wedge \\
&\mathsf{L}_{\mathsf{pos}}(0,0) \wedge \mathsf{L}_{\mathsf{pos}}(1,1) \wedge \mathsf{L}_{\mathsf{neg}}(0,1) \wedge \mathsf{L}_{\mathsf{neg}}(1,0) \rightarrow \exists \mathsf{y}.\mathsf{B}_1(\mathsf{y},1,\mathsf{x}_1) \wedge \mathsf{B}_1(\mathsf{y},0,\mathsf{x}_1) \\
&\qquad\qquad \wedge \ldots \wedge \mathsf{B}_\mathsf{k}(\mathsf{y},1,\mathsf{x}_\mathsf{k}) \wedge \mathsf{B}_\mathsf{k}(\mathsf{y},0,\mathsf{x}_\mathsf{k})
\end{aligned}$$

r_2:
$$\mathsf{B}_1(\mathsf{w},\mathsf{y}_1,\mathsf{y}_1) \wedge \ldots \wedge \mathsf{B}_\mathsf{k}(\mathsf{w},\mathsf{y}_\mathsf{k},\mathsf{y}_\mathsf{k}) \rightarrow \mathsf{Goal}$$

Facts matching the atom $\mathsf{V}(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k})$ in r_1 will be interpreted as truth assignments for $\vec{\mathsf{p}}$. Facts about N enumerate all cases in which a conjunction of three truth values evaluates to false, while facts about $\mathsf{L}_{\mathsf{pos}}$ and $\mathsf{L}_{\mathsf{neg}}$ yield a convenient way to compute the truth value of positive and negative propositional literals. The predicates B_i are used to constrain the possible truth assignments: the body of r_2 can only match the head of r_1 if each variable x_i is mapped to 0 or 1.

To define C, we consider variables $\xi_1, \ldots, \xi_\mathsf{k}$ (representing $\vec{\mathsf{p}}$), $\zeta_1, \ldots, \zeta_\mathsf{m}$ (representing $\vec{\mathsf{q}}$) and $\mathsf{u}_{11}, \mathsf{u}_{12}, \mathsf{u}_{13}, \ldots, \mathsf{u}_{\mathsf{n}1}, \mathsf{u}_{\mathsf{n}2}, \mathsf{u}_{\mathsf{n}3}$ (representing truth values of literals). Given any literal ℓ_{ij} of the QBF, let $\mathsf{L}_\mathsf{ij} = \mathsf{L}_{\mathsf{pos}}$ if ℓ_{ij} is a positive literal and $\mathsf{L}_\mathsf{ij} = \mathsf{L}_{\mathsf{neg}}$ otherwise, and let $\mathsf{v}_\mathsf{ij} = \xi_\mathsf{k}$ if ℓ_{ij} contains the existentially quantified propositional variable p_k and $\mathsf{v}_\mathsf{ij} = \zeta_\mathsf{k}$ if ℓ_{ij} contains the universally quantified propositional vari-

able q_k. We define C to consist of a single constraint c:

$$\begin{aligned} c: \quad & \mathsf{V}(\xi_1, \ldots, \xi_\mathsf{k}) \wedge \\ & \mathsf{N}(\mathsf{u}_{11}, \mathsf{u}_{12}, \mathsf{u}_{13}) \wedge \ldots \wedge \mathsf{N}(\mathsf{u}_{\mathsf{n}1}, \mathsf{u}_{\mathsf{n}2}, \mathsf{u}_{\mathsf{n}3}) \wedge \\ & \mathsf{L}_{11}(\mathsf{v}_{11}, \mathsf{u}_{11}) \wedge \mathsf{L}_{12}(\mathsf{v}_{12}, \mathsf{u}_{12}) \wedge \mathsf{L}_{13}(\mathsf{v}_{13}, \mathsf{u}_{13}) \wedge \\ & \qquad \ldots \\ & \mathsf{L}_{\mathsf{n}1}(\mathsf{v}_{\mathsf{n}1}, \mathsf{u}_{\mathsf{n}1}) \wedge \mathsf{L}_{\mathsf{n}2}(\mathsf{v}_{\mathsf{n}2}, \mathsf{u}_{\mathsf{n}2}) \wedge \mathsf{L}_{\mathsf{n}3}(\mathsf{v}_{\mathsf{n}3}, \mathsf{u}_{\mathsf{n}3}) \quad \rightarrow \bot \end{aligned}$$

First, we show $\Rightarrow$ of (6.1). Assume that $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ is satisfiable, i.e., there exists a truth assignment $v_\exists : \vec{\mathsf{p}} \rightarrow \{0,1\}$ such that for every truth assignment $v_\forall : \vec{\mathsf{q}} \rightarrow \{0,1\}$, we have $v_\exists \cup v_\forall \models \varphi$. We define a substitution θ by setting $\theta(\mathsf{x}_\mathsf{i}) = v_\exists(p_i)$ and a set of facts F as follows:

$$\begin{aligned} F = & \{\mathsf{V}(v_\exists(p_1), \ldots, v_\exists(p_k))\} \cup \\ & \{\mathsf{L}_\mathsf{pos}(0,0), \mathsf{L}_\mathsf{pos}(1,1), \mathsf{L}_\mathsf{neg}(0,1), \mathsf{L}_\mathsf{neg}(1,0)\} \cup \\ & \{\mathsf{N}(\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3) \mid \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \in \{0,1\}\} \setminus \{\mathsf{N}(1,1,1)\} \end{aligned}$$

It is easy to check that conditions (P1)–(P6) are satisfied. Suppose for a contradiction that $F \not\models c$ and let ψ denote the body of c. Then there exists a substitution σ for the variables in c, such that $\psi\sigma \subseteq F$. Since $\mathsf{V}(\xi_1, \ldots, \xi_\mathsf{k}) \in \psi$, we obtain $v_\exists(p_i) = \sigma(\xi_\mathsf{i})$ for all $\mathsf{i} = 1, \ldots, \mathsf{k}$. Similarly, define $v_\forall : \vec{\mathsf{q}} \rightarrow \{0,1\}$ by setting $v_\forall(q_i) = \sigma(\zeta_\mathsf{i})$. Clearly, the truth value for every literal ℓ_{ij} under $v_\exists \cup v_\forall$ is $\sigma(\mathsf{u}_\mathsf{ij})$. By $\psi\sigma \subseteq F$, we find $v_\exists \cup v_\forall \not\models \ell_{i1} \wedge \ell_{i2} \wedge \ell_{i3}$ for all $i = 1, \ldots, n$. Thus $v_\exists \cup v_\forall \not\models \varphi$, which contradicts the assumption on $v_\exists$. Thus $F \models c$ as required.

Next, we show $\Leftarrow$ of (6.1). Assume that $r_1 \xrightarrow{+}_C r_2$ is witnessed by a set of facts F and substitution θ. By (P1), $\mathsf{V}(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k})\theta \in F$ and, by (P3) and (P5), we have $\mathsf{B}_\mathsf{i}(\mathsf{w}, \mathsf{y}_\mathsf{i}, \mathsf{y}_\mathsf{i})\theta \in \{\mathsf{B}_\mathsf{i}(\mathsf{f}(\vec{\mathsf{x}}), 1, \mathsf{x}_\mathsf{i})\theta, \mathsf{B}_\mathsf{i}(\mathsf{f}(\vec{\mathsf{x}}), 0, \mathsf{x}_\mathsf{i})\theta\}$ for every $\mathsf{i} \in \{1, \ldots, \mathsf{k}\}$, where f is the skolem function used when skolemising r_1. This implies that $\theta(\mathsf{x}_\mathsf{i}) \in \{0,1\}$ for all $\mathsf{i} \in \{1, \ldots, \mathsf{k}\}$. We define $v_\exists : \vec{\mathsf{p}} \rightarrow \{0,1\}$ by setting $v_\exists(p_i) = \theta(\mathsf{x}_\mathsf{i})$. Suppose for a contradiction that there is a truth assignment $v_\forall : \vec{\mathsf{q}} \rightarrow \{0,1\}$ such that $v_\exists \cup v_\forall \not\models \varphi$. Let σ be the substitution for the variables in c defined by setting $\sigma(\xi_\mathsf{i}) = v_\exists(p_i)$, $\sigma(\zeta_\mathsf{i}) = v_\forall(q_i)$, and $\sigma(\mathsf{u}_\mathsf{ij}) = 1$ if $v_\exists \cup v_\forall \models \ell_{ij}$ and $\sigma(\mathsf{u}_\mathsf{ij}) = 0$ otherwise. It is easy to

see that $\psi\sigma \subseteq F$, where ψ is the body of c as above. This contradicts $r_1 \xrightarrow{+}_C r_2$, so we conclude that no such $v_\forall$ exists, i.e., that $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ is satisfiable. □

Similarly to the case without constraints, checking negative reliance under a set of constraints is less computationally hard than checking positive reliance under constraints: in Theorem 85 we prove that testing whether one rule negatively relies on another one under a set of constraints is in Δ_2^P.

Theorem 81. *Given rules r_1 and r_2 and a set of constraints C, the problem of deciding whether $r_1 \xrightarrow{-}_C r_2$ is in Δ_2^P.*

Proof. In the proof of Theorem 66, we showed that $r_1 \xrightarrow{-} r_2$ can be decided in polynomial time by describing a polynomial, correct and complete algorithm. We extend the same algorithm to check $r_1 \xrightarrow{-}_C r_2$, by checking for each F and θ such that $r_1 \xrightarrow{-} r_2$ (F and θ as defined in the correctness part of Theorem 66 proof) whether $F \models C$. In order to decide whether $F \models C$, we need to check for each $c \in C$ whether $F \models c$, which requires a call to an NP-oracle. Since the number of possible F and θ pairs that the algorithm tests is polynomial in the size of the input, the algorithm needs at most a polynomial number of steps, each of which invokes a polynomial number of calls to an NP-oracle. As a consequence, a polynomial number of calls to an NP-oracle is required overall and the problem lies in Δ_2^P. □

The lower bound for checking negative reliance under constraints is left open, as the complexities for checking both R-acyclicity and R-stratification are dominated by the complexity for checking positive reliance under constraints, for which tight bounds have been proved.

6.2 R-Acyclicity under Constraints

Similarly to checking R-acyclicity without constraints, the complexity of deciding R-acyclicity given a set of constraints is dominated by the complexity of testing positive reliances under these constraints. We thus show next that determining whether a program is R-acyclic under constraints is Π_2^P-complete.

Theorem 82. *Let P be a program and let C be a set of constraints. Checking whether P is R-acyclic under C is Π_2^P-complete.*

Proof. Membership follows since one can check in Σ_2^P that P is *not* R-acyclic under constraints, analogously to the case of R-acyclicity in Theorem 41. For hardness, consider the rules r_1 and r_2 and constraints C as constructed in the hardness proof of Theorem 80, as well as the rule $r_3 : \mathsf{Goal} \rightarrow \mathsf{Start}$. Let r_1' be obtained from r_1 by adding Start to its body atoms. Clearly, the program $\{r_1', r_2, r_3\}$ is R-acyclic under C if and only if $\exists \vec{\mathsf{p}}.\forall \vec{\mathsf{q}}.\varphi$ is not satisfiable. □

Examples 77 and 79 already shows that R-stratification under constraints strictly extends R-stratification. The following example, where rules are used to represent properties of pharmaceuticals, demonstrates that also the class of R-acyclic programs under constraints strictly extends the class of R-acyclic programs.

Example 83. Consider the following program P consisting of rules r_1, r_2, r_3 and r_4 modelling knowledge about drugs. The example is motivated by the extensive use of has role relationships within ChEBI ontology (more than 25,000 as of November 2012 [111]), where such expressions are used to describe biochemical roles of molecular structures. The classes analgesic (CHEBI id 35480), opioid analgesic (CHEBI id 35482) and non-narcotic analgesic (CHEBI id 35481) are used to characterise drugs with a pain relieving effect and whose mechanism of action involves binding to an opioid receptor or not, respectively. One can represent this knowledge, which appears in natural language definitions within ChEBI, with rules r_1–r_4.

$$r_1 : \quad \mathsf{analgesicDrug}(\mathsf{x}) \rightarrow \exists \mathsf{y}.\mathsf{hasRole}(\mathsf{x},\mathsf{y}) \wedge \mathsf{analgesic}(\mathsf{y})$$

$$r_2 : \mathsf{hasRole}(\mathsf{x},\mathsf{z}) \wedge \mathsf{analgesic}(\mathsf{z}) \wedge \ \mathbf{not}\ \mathsf{opioid}(\mathsf{x}) \rightarrow \mathsf{hasNonNarcoticAnalgesicRole}(\mathsf{x},\mathsf{z})$$

$$r_3 : \quad \mathsf{hasNonNarcoticAnalgesicRole}(\mathsf{x}_1,\mathsf{x}_2) \rightarrow \mathsf{hasAnalgesicRole}(\mathsf{x}_1,\mathsf{x}_2)$$

$$r_4 : \quad \mathsf{opioid}(\mathsf{x}) \wedge \mathsf{hasAnalgesicRole}(\mathsf{x},\mathsf{z}) \rightarrow \mathsf{opioidDrug}(\mathsf{x}) \wedge \mathsf{analgesicDrug}(\mathsf{x})$$

One can verify that $r_1 \xrightarrow{+} r_2 \xrightarrow{+} r_3 \xrightarrow{+} r_4 \xrightarrow{+} r_1$ which implies that P is not R-acyclic. Yet, for sets of facts $F_1 = \{\mathsf{analgesicDrug}(\mathsf{a})\}$ and $F_2 = \{\mathsf{analgesicDrug}(\mathsf{a}), \mathsf{opioid}(\mathsf{a})\}$ both programs $P \cup F_1$ and $P \cup F_2$ have a finite stable model. Consider now the

following constraint c that yields a contradiction every time that an opioid is linked to a non-narcotic analgesic role.

$$c : \mathsf{opioid}(\mathsf{x}) \wedge \mathsf{nonNarcoticAnalgesicRole}(\mathsf{x}, \mathsf{y}) \rightarrow \bot$$

We have that r_4 no longer positively relies on r_3 under $\{c\}$ which results in P being R-acyclic under the constraint above. ♢

Theorem 45 can be generalised to programs that are R-acyclic under constraints:

Theorem 84. *For a set of facts F, a fact α and a program P that is R-acyclic under a set of constraints C, deciding $P \cup F \cup C \models \alpha$ is* coN2ExpTime*-complete w.r.t. program complexity and* coNP*-complete w.r.t. data complexity.*

Proof. Since extending P with constraints does not increase the bound on the number of terms and checking satisfaction of constraints is in NP w.r.t. program complexity and polynomial w.r.t. data complexity, the claims follows from Theorem 45. □

6.3 R-Stratification under Constraints

Similarly to the case for $\xrightarrow{+}$ and $\xrightarrow{-}$, the relations $\xrightarrow{+}_C$ and $\xrightarrow{-}_C$ induce a graph of reliances under constraints. Analogously to Proposition 65, we can show that P is R-stratified under constraints if and only if this graph does not contain cycles that involve $\xrightarrow{-}_C$. This is the basis for deciding R-stratification under constraints, leading to the following result. The proof of Theorem 85 is similar in spirit to the proof of Theorem 66: the complexity results for checking reliances serve as the base for checking R-stratification under a set of constraints.

Theorem 85. *Checking whether a program P is R-stratified under C is Π_2^P-complete.*

Proof. Membership follows by the fact that it can be checked in Σ_2^P whether a program P is *not* R-stratified under a set of constraints C. Indeed, as discussed above, this can be detected by finding a cycle of the form $r_0 \xrightarrow{\pm}_C \ldots \xrightarrow{\pm}_C r_k \xrightarrow{-}_C r_{k+1} \xrightarrow{\pm}_C \ldots \xrightarrow{\pm}_C r_{\ell-1} \xrightarrow{\pm}_C r_0$, where $\ell \leq |P|$. One can guess such a cycle and justifications

F_i, θ_i for each of the reliances $r_i \stackrel{+}{\Rightarrow}_C r_{(i+1) \bmod \ell}$ and these choices can be verified by a polynomial number of calls to an NP-oracle.

For hardness, we show that checking satisfiability of a quantified Boolean formula $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ can be reduced to checking whether a program P is *not* R-stratified under a set of constraints C. In the proof of Theorem 80, we constructed a set of constraints C and rules $r_1 : B_1 \to H_1$ and $r_2 : B_2 \to \mathsf{Goal}$ satisfying (6.1). If we modify r_1 to $r_1' : B_1 \wedge$ **not** $\mathsf{Goal} \to H_1$, then $\exists\vec{\mathsf{p}}.\forall\vec{\mathsf{q}}.\varphi$ is satisfiable if and only if $\{r_1', r_2\}$ is not R-stratified under C, which proves our claim. □

Given an R-stratification of P under a set of constraints C, we can again define a computation scheme to obtain unique stable models. C in this case is evaluated on all strata, though one can also defer constraint checking to the highest stratum.

Definition 86. *For a set of facts F and a program P with R-stratification $\vec{\mathsf{P}} = P_1, \ldots, P_n$ under constraints C, define $S^0_{\vec{\mathsf{P}},C}(F) = T_C(F)$ and*

$$S^{i+1}_{\vec{\mathsf{P}},C}(F) = T^{\infty}_{P_{i+1}\cup C}(S^i_{\vec{\mathsf{P}},C}(F)) \quad \textit{for} \quad 0 \leq i < n.$$

In the rest of the section, we abbreviate $S^i_{\vec{\mathsf{P}},C}(F)$ with $S^i_{\vec{\mathsf{P}},C}$.

The following result can be shown using the same overall proof structure as in Section 5.2. The main difference is that in all arguments that discuss potential reliances between rules, we also need to show satisfaction of the constraints. This is usually a consequence of the assumption that $\bot$ is not derived.

Theorem 87. *If $\bot \notin S^n_{\vec{\mathsf{P}},C}(F)$, then $S^n_{\vec{\mathsf{P}},C}(F)$ is the unique stable model of $F \cup P \cup C$ or else $F \cup P \cup C$ has no stable model.*

We first show some intermediate results that we will use for proving the main claim. Lemmas 88 and 89 are analogous to lemmas 69 and 70; the only difference is that we assume satisfaction of constraints C after having completed the saturation of the stratum.

Lemma 88. *Consider numbers $1 \leq i \leq j \leq k \leq n$ and $\ell \geq 0$, a rule $r \in P_i$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a substitution θ such that $\bot \notin S^k_{\vec{\mathsf{P}},C}$. It holds that $T^{\ell}_{P_j\cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \models B^+\theta$, **not** $B^-\theta$ implies $S^k_{\vec{\mathsf{P}},C} \models B^+\theta$, **not** $B^-\theta$.*

Proof. To simplify the presentation, define $\mathcal{M} = T^{\ell}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C})$ and $\mathcal{M}' = S^{k}_{\vec{\mathsf{P}},C}$. We assume $\mathcal{M} \models B^+\theta,\ \textbf{not}\ B^-\theta$; suppose for a contradiction that $\mathcal{M}' \not\models B^+\theta,\ \textbf{not}\ B^-\theta$. Since $\mathcal{M} \subseteq \mathcal{M}'$, we find that $\mathcal{M}' \models B^+\theta$. Hence $\mathcal{M}' \not\models \textbf{not}\ B^-\theta$, that is, $\mathcal{M}' \cap B^-\theta \neq \emptyset$. Thus there are $m \geq j$ and $o \geq 0$ such that $T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \cap B^-\theta = \emptyset$ and $T^{o+1}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \cap B^-\theta \neq \emptyset$. So there is a rule $r_1 \in P_m$ with $\mathsf{sk}(r_1) = (B^+_1, B^-_1, H_1)$ and a substitution θ_1 such that

$$B^+_1\theta_1 \subseteq T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \tag{6.2}$$

$$B^-_1\theta_1 \cap T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) = \emptyset \tag{6.3}$$

$$H_1\theta_1 \cap B^-\theta \neq \emptyset \tag{6.4}$$

We show that $r_1 \xrightarrow{-}_C r$. For brevity, let $F' = T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C})$. We demonstrate that $G = \gamma_{F'}(T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}))$ and $\sigma = \gamma_{F'}(\theta_1 \cup \theta)$ establish the conditions for $r_1 \xrightarrow{-}_C r$ in Definition 59:

(N1) $B^+_1\sigma \subseteq G$ by (6.2).

(N2) $B^-_1\sigma \cap G = \emptyset$ by (6.3).

(N3) $B^+\sigma \subseteq G$ follows from $T^{\ell}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \models B^+\theta$ and $T^{\ell}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \subseteq T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C})$; note that if $m = j$, then $\ell \leq o$ which follows from $T^{\ell}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \cap B^-\theta = \emptyset$ and $T^{o+1}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \cap B^-\theta \neq \emptyset$.

(N4) $B^-\sigma \cap H_1\sigma \neq \emptyset$ by (6.4); note that for every skolem term $\mathsf{f}(\vec{\mathsf{x}})\theta_1$ in $H_1\theta_1$, the definition of $\gamma_{F'}$ ensures $\gamma_{F'}(\mathsf{f}(\vec{\mathsf{x}})\theta_1) = \mathsf{f}(\gamma_{F'}(\vec{\mathsf{x}}\theta_1))$ (the former occurs in $B^-\sigma'$, the latter occurs in $H_1\sigma'$).

(N5) $B^-\sigma \cap G = \emptyset$ by $T^{o}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \cap B^-\theta = \emptyset$.

Since $\bot \notin G$, we have $G \models C$. Thus $r_1 \xrightarrow{-}_C r$. However, $r \in P_i$, $r_1 \in P_m$ and $i \leq j \leq m$, which contradict Definition 61. □

Lemma 89. *Consider numbers $1 \leq i < j \leq k \leq n$, a rule $r \in P_i$ and a substitution θ such that $\bot \notin S^{k}_{\vec{\mathsf{P}},C}$. Then $S^{j}_{\vec{\mathsf{P}},C} \models \mathsf{sk}(r)\theta$ implies $S^{k}_{\vec{\mathsf{P}},C} \models \mathsf{sk}(r)\theta$.*

Proof. Let $\mathsf{sk}(r) = (B^+, B^-, H)$ and suppose for a contradiction that $S^j_{\vec{\mathsf{P}},C} \models \mathsf{sk}(r)\theta$ and $S^k_{\vec{\mathsf{P}},C} \not\models \mathsf{sk}(r)\theta$. Since $S^j_{\vec{\mathsf{P}},C} \subseteq S^k_{\vec{\mathsf{P}},C}$, we cannot have either $B^-\theta \cap S^j_{\vec{\mathsf{P}},C} \neq \emptyset$ or $S^j_{\vec{\mathsf{P}},C} \models (B^+ \cup H)\theta$, **not** $B^-\theta$. So, $B^+\theta \not\subseteq S^j_{\vec{\mathsf{P}},C}$. By $S^k_{\vec{\mathsf{P}},C} \not\models \mathsf{sk}(r)\theta$, we derive $S^k_{\vec{\mathsf{P}},C} \models B^+\theta$, **not** $B^-\theta$ and $S^k_{\vec{\mathsf{P}},C} \not\models H\theta$. Therefore, there exists a maximal non-empty set of facts A such that $A \subseteq B^+\theta$ and $A \cap S^j_{\vec{\mathsf{P}},C} = \emptyset$. Hence there are numbers $\ell \geq 0$ and m such that $j < m \leq k$, a rule $r_1 \in P_m$ with $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and a substitution θ_1 such that

$$B_1^+\theta_1 \subseteq T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \tag{6.5}$$

$$B_1^-\theta_1 \cap T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) = \emptyset \tag{6.6}$$

$$H_1\theta_1 \cap A \not\subseteq T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \tag{6.7}$$

Let $F' = T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \cup (A \setminus H_1\theta_1)$, let $G = \gamma_{F'}(T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \cup (A \setminus H_1\theta_1))$ and let $\sigma = \gamma_{F'}(\theta_1 \cup \theta)$. We show that G and σ meet the conditions for $r_1 \xrightarrow{\pm}_C r$ in Definition 39.

(P1) $B_1^+\sigma \subseteq G$ by (6.5).

(P2) $B_1^-\sigma \cap G = \emptyset$ by (6.6) and $B_1^-\theta_1 \cap A = \emptyset$; note that the latter follows from $A \subseteq S^k_{\vec{\mathsf{P}},C}$ and $S^k_{\vec{\mathsf{P}},C} \models B_1^+\theta_1$, **not** $B_1^-\theta_1$ which is a consequence of Lemma 88, $T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \models B_1^+\theta_1$, **not** $B_1^-\theta_1$, $r_1 \in P_m$ and $k \geq m$.

(P3) $B^+\sigma \subseteq G \cup H_1\sigma$ by $B^+\theta \subseteq F' \cup H_1\theta_1$.

(P4) $B^-\sigma \cap (G \cup H_1\sigma) = \emptyset$ by $S^k_{\vec{\mathsf{P}},C} \models B^+\theta$, **not** $B^-\theta$.

(P5) $B^+\sigma \not\subseteq G$ by (6.7).

(P6) $H\sigma \not\subseteq G \cup H_1\sigma$ by $G \cup H_1\theta_1 \subseteq S^k_{\vec{\mathsf{P}},C}$ and $S^k_{\vec{\mathsf{P}},C} \not\models H\theta$.

Also $\bot \notin G$, so, $G \models C$. Thus, by $r_1 \in P_m$, $r \in P_i$ and $m > i$ we derive a contradiction and show our initial claim. □

Proposition 90 is analogous to Proposition 71; the only difference is that we now refer to a set of derived facts that also satisfy the constraints in C.

Proposition 90. *If* $\bot \notin S^n_{\vec{\mathsf{P}},C}$, *then* $S^n_{\vec{\mathsf{P}},C} \models_{\mathsf{SM}} F \cup P \cup C$.

Proof. We show that $\bot \notin S^n_{\vec{\mathsf{P}},C}$ implies (♣) $S^n_{\vec{\mathsf{P}},C}$ is a model of $F \cup \mathsf{GL}(P^n_1 \cup C, S^n_{\vec{\mathsf{P}},C})$ and (♠) the model $S^n_{\vec{\mathsf{P}},C}$ of $F \cup \mathsf{GL}(P^n_1 \cup C, S^n_{\vec{\mathsf{P}},C})$ is minimal.

(♣) We prove $S^k_{\vec{\mathsf{P}},C} \models_{\mathsf{SM}} F \cup \mathsf{GL}(P^k_1 \cup C, S^k_{\vec{\mathsf{P}},C})$ for all $k \in \{0, \ldots, n\}$ by induction over k. If $k = 0$, then $P^k_1 = \emptyset$ and $S^k_{\vec{\mathsf{P}},C} = F$ (since $\bot \notin S^n_{\vec{\mathsf{P}},C}$). Clearly, F is a stable model of $F \cup \mathsf{GL}(C, F)$, since $T^\infty_{\mathsf{GL}(C,F)}(F) = F$.

For the induction step, let $S^k_{\vec{\mathsf{P}},C}$ be a model of $F \cup \mathsf{GL}(P^k_1 \cup C, S^k_{\vec{\mathsf{P}},C})$ (induction hypothesis). We claim that $S^{k+1}_{\vec{\mathsf{P}},C}$ is a model of $F \cup \mathsf{GL}(P^{k+1}_1 \cup C, S^{k+1}_{\vec{\mathsf{P}},C})$. Note that $S^{k+1}_{\vec{\mathsf{P}},C} \models F \cup \mathsf{GL}(P^{k+1}_1 \cup C, S^{k+1}_{\vec{\mathsf{P}},C})$ is equivalent to $S^{k+1}_{\vec{\mathsf{P}},C} \models F \cup \mathsf{sk}(P^{k+1}_1 \cup C)$. Since $S^{k+1}_{\vec{\mathsf{P}},C} = T^\infty_{P_{k+1} \cup C}(S^k_{\vec{\mathsf{P}},C})$, we find $S^{k+1}_{\vec{\mathsf{P}},C} \models F \cup \mathsf{sk}(P_{k+1})$. It remains to show $S^{k+1}_{\vec{\mathsf{P}},C} \models \mathsf{sk}(P^k_1 \cup C)$. Since $\bot \notin S^n_{\vec{\mathsf{P}},C}$, we have $\bot \notin S^{k+1}_{\vec{\mathsf{P}},C}$. Moreover, consider an arbitrary rule $r \in P^k_1$. By induction hypothesis, $S^k_{\vec{\mathsf{P}},C} \models \mathsf{sk}(P^k_1 \cup C)$ and thus $S^k_{\vec{\mathsf{P}},C} \models \mathsf{sk}(r)$. By Lemma 89, $S^{k+1}_{\vec{\mathsf{P}},C} \models \mathsf{sk}(r)$. Since r was arbitrary, this establishes the claim.

(♠) Suppose for a contradiction that there is $\mathcal{M} \subsetneq S^n_{\vec{\mathsf{P}},C}$ such that it satisfies $\mathcal{M} \models F \cup \mathsf{GL}(P^n_1 \cup C, S^n_{\vec{\mathsf{P}},C})$. Then, there are $\ell \geq 0$ and j, where $1 \leq j \leq k$, such that $T^\ell_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \subseteq \mathcal{M}$ and $T^{\ell+1}_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \not\subseteq \mathcal{M}$. Thus, there is a rule $r \in P_j$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a substitution θ with

$$B^+\theta \subseteq T^\ell_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \tag{6.8}$$

$$B^-\theta \cap T^\ell_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) = \emptyset \tag{6.9}$$

$$H\theta \not\subseteq \mathcal{M} \tag{6.10}$$

By $T^\ell_{P_j \cup C}(S^{j-1}_{\vec{\mathsf{P}},C}) \subseteq \mathcal{M}$ and (6.8), $B^+\theta \subseteq \mathcal{M}$; also due to $\mathcal{M} \models F \cup \mathsf{GL}(P^n_1 \cup C, S^n_{\vec{\mathsf{P}},C})$ and (6.10), it follows that $(B^+\theta, \emptyset, H\theta) \notin \mathsf{GL}(P^n_1 \cup C, S^n_{\vec{\mathsf{P}},C})$ and thus $B^-\theta \cap S^n_{\vec{\mathsf{P}},C} \neq \emptyset$. However, by (6.8), (6.9) and Lemma 88, $S^n_{\vec{\mathsf{P}},C} \models B^+\theta$, **not** $B^-\theta$, which yields the required contradiction. □

Lemma 91 is analogous to lemma 72. The difference is that in this case the stable model coincides with the computed set of facts $S^n_{\vec{\mathsf{P}},C}$, which have the additional property of satisfying the constraints in C.

Lemma 91. *If* $\mathcal{M} \models_{\mathsf{SM}} F \cup P \cup C$, *then* $S^n_{\vec{\mathsf{P}},C} = \mathcal{M}$.

Proof. We inductively show that for each $k \in \{0, \ldots, n\}$, it is $S^k_{\vec{\mathsf{P}},C} = T^\infty_{\mathsf{GL}(P_1^k \cup C, \mathcal{M})}(F)$. This establishes the claim since the latter is equal to $\mathcal{M}$ if $k = n$.

For $k = 0$, it is $S^0_{\vec{\mathsf{P}},C} = F$ and $T^\infty_{\mathsf{GL}(P_1^0 \cup C, \mathcal{M})}(F) = C(F)$. Since $\mathcal{M} \models_{\mathsf{SM}} F \cup P \cup C$, we have $\bot \notin C(F)$.

For the induction step, assume that the claim has been shown for k. We show that $S^{k+1}_{\vec{\mathsf{P}},C} = T^\infty_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F)$.

We first show that $S^{k+1}_{\vec{\mathsf{P}},C} \subseteq T^\infty_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F)$. For brevity, we use T^i to denote $T^\infty_{\mathsf{GL}(P_1^i \cup C, \mathcal{M})}(F)$. Suppose for a contradiction that $S^{k+1}_{\vec{\mathsf{P}},C} \not\subseteq T^{k+1}$. Then there are numbers $m \in \{1, \ldots, k+1\}$ and $\ell \geq 0$ such that

$$T^\ell_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \subseteq T^{k+1} \tag{6.11}$$

$$T^{\ell+1}_{P_m \cup C}(S^{m-1}_{\vec{\mathsf{P}},C}) \not\subseteq T^{k+1} \tag{6.12}$$

By the induction hypothesis, $S^k_{\vec{\mathsf{P}},C} \subseteq T^{k+1}$, so (6.12) implies $m = k+1$. Thus there is a rule $r \in P_{k+1}$ with $\mathsf{sk}(r) = (B^+, B^-, H)$ and a substitution θ such that

$$B^+\theta \subseteq T^\ell_{P_{k+1} \cup C}(S^k_{\vec{\mathsf{P}},C}) \tag{6.13}$$

$$B^-\theta \cap T^\ell_{P_{k+1} \cup C}(S^k_{\vec{\mathsf{P}},C}) = \emptyset \tag{6.14}$$

$$H\theta \not\subseteq T^{k+1} \tag{6.15}$$

Together, (6.13), (6.11) (where $m = k+1$) and (6.15) imply

$$B^-\theta \cap T^{k+1} \neq \emptyset \tag{6.16}$$

Thus there is a rule r_1 with $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$, a substitution θ_1 and a number j such that

$$B_1^+\theta_1 \subseteq T^j_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F) \tag{6.17}$$

$$B_1^-\theta_1 \cap \mathcal{M} = \emptyset \tag{6.18}$$

$$B^-\theta \cap T^j_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F) = \emptyset \tag{6.19}$$

$$B^-\theta \cap H_1\theta_1 \neq \emptyset \tag{6.20}$$

Define $F' = T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \cup T^\ell_{P_{k+1}\cup C}(S^k_{\vec{\mathsf{P}},C})$. We claim that the set of facts $G = \gamma_{F'}(F')$ and the substitution $\sigma = \gamma_{F'}(\theta \cup \theta_1)$ satisfy the conditions for $r_1 \xrightarrow{-}_C r$ in Definition 59. By definition, G does not contain function symbols.

(N1) $B_1^+\sigma \subseteq G$ by (6.17).

(N2) $B_1^-\sigma \cap G = \emptyset$ by (6.18) and $F' \subseteq \mathcal{M}$, obtained from (6.11) (where $m = k+1$).

(N3) $B^+\sigma \subseteq G$ by (6.13).

(N4) $B^-\sigma \cap H_1\sigma \neq \emptyset$ by (6.20).

(N5) $B^-\sigma \cap G = \emptyset$ by (6.14) and (6.19).

By $\mathcal{M} \models F \cup P \cup C$ and $T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \subseteq \mathcal{M}$, it is $\bot \notin T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$; by (6.11) (where $m = k+1$) we have $T^\ell_{P_{k+1}\cup C}(S^k_{\vec{\mathsf{P}},C}) \subseteq T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ and, so, $\bot \notin T^\ell_{P_{k+1}\cup C}(S^k_{\vec{\mathsf{P}},C})$. Therefore, $\bot \notin F'$ and, thus, $\bot \notin G$, which implies $G \models C$. As a consequence, $r_1 \xrightarrow{-} r$ and therefore $r_1 \in P_1^k$ by Definition 61. By (6.18) and $T^k \subseteq \mathcal{M}$, we have $B_1^-\theta_1 \cap T^k = \emptyset$. By the induction hypothesis, $T^k = S^k_{\vec{\mathsf{P}},C}$, so $B_1^-\theta_1 \cap S^k_{\vec{\mathsf{P}},C} = \emptyset$. By (6.20) and (6.14), $H_1\theta_1 \not\subseteq S^k_{\vec{\mathsf{P}},C}$. Together, these observations imply that $B_1^+\theta_1 \not\subseteq S^k_{\vec{\mathsf{P}},C}$. Using again $T^k = S^k_{\vec{\mathsf{P}},C}$, we find that $B_1^+\theta_1 \not\subseteq T^k$.

From $B_1^+\theta_1 \not\subseteq T^k$ and (6.17), we conclude that there is a rule $r_2 \in P_{k+1}$ with $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$, a substitution θ_2 and a number $o < j$ such that

$$B_2^+\theta_2 \subseteq T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \tag{6.21}$$

$$B_2^-\theta_2 \cap \mathcal{M} = \emptyset \tag{6.22}$$

$$H_2\theta_2 \cap B_1^+\theta_1 \not\subseteq T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \tag{6.23}$$

By (6.19) and (6.20), there is $\alpha \in H_1\theta_1 \setminus T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ and by (6.23), there is $\beta \in (H_2\theta_2 \cap B_1^+\theta_1) \setminus T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$. Let $F'' = \mathcal{M} \setminus \{\alpha, \beta\}$. We claim that the set of facts $G' = \gamma_{F''}(F'')$ and substitution $\sigma' = \gamma_{F''}(\theta_1 \cup \theta_2)$ satisfy the conditions for $r_2 \xrightarrow{+}_C r_1$ in Definition 39. By definition, G' does not contain function symbols.

(P1) $B_2^+\sigma' \subseteq G'$ by (6.21) and $T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \subseteq F''$. For the latter, note that $\beta \notin T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ and $\alpha \notin T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ since $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ and $T^o_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F) \subseteq T^j_{\mathsf{GL}(P_1^{k+1}\cup C,\mathcal{M})}(F)$ (by $o < j$).

(P2) $B_2^- \sigma' \cap G' = \emptyset$ by (6.22).

(P3) $B_1^+ \sigma' \subseteq G' \cup H_2 \sigma'$ by (6.17) and $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F)$.

(P4) $B_1^- \sigma' \cap (G' \cup H_2 \sigma') = \emptyset$ by (6.18) and $H_2 \theta_2 \subseteq \mathcal{M}$.

(P5) $B_1^+ \sigma' \not\subseteq G'$ by our choice of β.

(P6) $H_1 \sigma' \not\subseteq G' \cup H_2 \sigma'$ is a consequence of our choice of α and $\alpha \notin T^j_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F)$.

Since $\mathcal{M} \models_{\mathsf{SM}} F \cup P \cup C$, we have $\bot \notin F''$ and, so, $G' \models C$. As a consequence, $r_2 \xrightarrow{+}_C r_1$. This contradicts the assumption that $r_1 \in P_1^k$ while $r_2 \in P_{k+1}$, and thus refutes our initial assumption that $S^{k+1}_{\vec{\mathsf{P}},C} \not\subseteq T^{k+1}$. We have thus shown that $S^{k+1}_{\vec{\mathsf{P}},C} \subseteq T^{k+1}$.

By $S^{k+1}_{\vec{\mathsf{P}},C} \subseteq T^{k+1}$, we have $\bot \notin S^{k+1}_{\vec{\mathsf{P}},C}$. The latter combined with Proposition 90 yields $S^{k+1}_{\vec{\mathsf{P}},C} \models_{\mathsf{SM}} F \cup P_1^{k+1} \cup C$. Thus $S^{k+1}_{\vec{\mathsf{P}},C} = T^\infty_{\mathsf{GL}(P_1^{k+1} \cup C, S^{k+1}_{\vec{\mathsf{P}},C})}(F)$. Now since $S^{k+1}_{\vec{\mathsf{P}},C} \subseteq T^{k+1} \subseteq \mathcal{M}$, we find that

$$\mathsf{GL}(P_1^{k+1} \cup C, S^{k+1}_{\vec{\mathsf{P}},C}) \supseteq \mathsf{GL}(P_1^{k+1} \cup C, T^{k+1}) \supseteq \mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M}).$$

So, $S^{k+1}_{\vec{\mathsf{P}},C} = T^\infty_{\mathsf{GL}(P_1^{k+1} \cup C, S^{k+1}_{\vec{\mathsf{P}},C})}(F) \supseteq T^\infty_{\mathsf{GL}(P_1^{k+1} \cup C, \mathcal{M})}(F) = T^{k+1}$ as required. □

Proof of Theorem 87. If $\bot \notin S^n_{\vec{\mathsf{P}},C}$, then by Proposition 90, $S^n_{\vec{\mathsf{P}},C} \models_{\mathsf{SM}} F \cup P \cup C$, which together with Lemma 91 implies that $S^n_{\vec{\mathsf{P}},C}$ is the unique stable model.

If $\bot \in S^n_{\vec{\mathsf{P}},C}$ assume for a contradiction that there exists set of facts $\mathcal{M}$, such that $\mathcal{M} \models_{\mathsf{SM}} F \cup P \cup C$. By Lemma 91, $\mathcal{M} = S^n_{\vec{\mathsf{P}},C}$ which is not possible since no stable model may contain bottom. □

Theorem 74 can be extended to programs that are R-acyclic, R-stratified under constraints:

Theorem 92. *For a set of facts F, a fact α and a program P that is R-acyclic and R-stratified under a set of constraints C, deciding $P \cup F \cup C \models \alpha$ is* 2ExpTime*-complete w.r.t. program complexity and* P*-complete w.r.t. data complexity.*

Proof. By an argument similar to the one used in the proof of Theorem 84, the claims follow from Theorem 74. □

6.4 Complexity Results for Reliances and Discussion

We conclude this chapter by providing an overview of the complexity results for reliance-related decision problems and a short discussion on related work.

Condition to check	**No constraints**	**Under constraints**
Positive reliance	NP-complete (Theorem 41)	Σ_2^P-complete (Theorem 80)
Negative reliance	P (Theorem 66)	Δ_2^P (Theorem 85)
R-acyclicity/R-stratification	coNP-complete (Theorems 44 and 66)	Π_2^P-complete (Theorems 82 and 85)

Table 6.1: Complexity of checking reliances, R-acyclicity and R-stratification

Table 6.1 illustrates complexity bounds for checking positive and negative reliances as well as R-acyclicity and R-stratification. We observe that checking positive reliances comes at a higher cost than negative (unless P = NP) which is intuitively explained as follows. For the positive case one needs to check the existence of a matching from an atom of the positive body of a rule, to the head of another, where that head may contain skolem terms; if skolem terms are present, then since the witness set of facts is not allowed to contain any skolem terms the matching needs to be extended so that it also covers atoms of the positive body that share variables with the initially considered positive body atom. In other words, a graph homomorphism check is needed in the worst case. However, for the negative case it is sufficient to identify just *one* atom whose predicate occurs in both the negative body and the head and subsequently perform a linear time unification check; the head atom may not contain skolem terms, since that would contradict the safety condition and (N3), according to which the witness set of facts is required to satisfy the positive body of the second rule. Since the number of atoms that have predicates in common is bounded by the size of the two rules, the check for negative reliance remains polynomial.

On the other hand, the derived complexity bounds for testing R-acyclicity and R-stratification coincide because both conditions require positive reliance checks. At

Type of program for fact entailment	**Data complexity**	**Combined complexity**
R-acyclic (no/under constraints)	coNP-complete (Theorems 45 and 84)	coN2ExpTime-complete (Theorems 45 and 84)
R-acyclic and R-stratified (no/under constraints)	P-complete (Theorems 74 and 92)	2ExpTime-complete (Theorems 74 and 92)

Table 6.2: Complexity of reasoning for decidable classes of nonmonotonic existential rules

the same time, testing reliances under constraints is computationally harder (unless the polynomial hierarchy collapses), which is justified by the additional check of constraints satisfaction that needs to be performed after identifying a witness set of facts and substitution. Given that the constraints check is equivalent to solving an instance of the graph homomorphism problem, which is NP-complete, a polynomial number of calls to an NP-oracle is required, which accounts for the complexity rise.

Table 6.2 depicts the complexity of reasoning for fragments of nonmonotonic existential rules that are defined using reliances. The drop in computational complexity is a straightforward consequence of the unique stable model property exhibited by programs with an R-stratification.

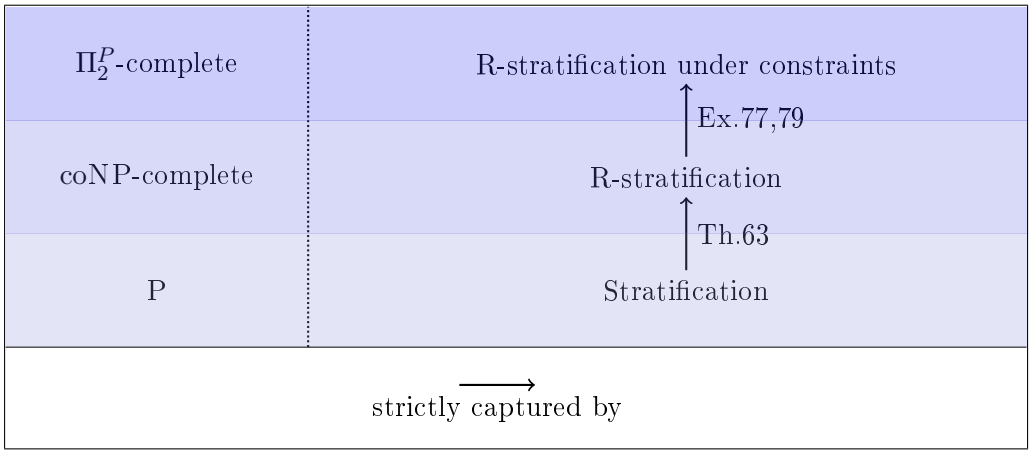


Figure 6.1: Comparison of stratification conditions w.r.t. (combined) complexity of checking and generality

Figure 6.1 illustrates the 'stratification zoo' for some of the stratification conditions presented here, similarly to the 'acyclicity zoo' of Section 4.3.

Regarding related work, this is not the first time that constraints are exploited to achieve more efficient reasoning. Ross [211] introduces a stratification condition, called *universal constraint stratification*, that extends local stratification by requiring the satisfaction of a set of *monotonicity constraints* by a set of facts, which—in his setting—is a relational database. Monotonicity constraints [36] are constraints different from the ones considered here; monotonicity constraints require the existence of a strict partial order on the set of database constants, such that the order complies with a set of inequalities specified w.r.t. certain predicates and rules. Ross proves that universal constraint satisfaction extends local stratification and can be efficiently recognised; however, he only considers function-free rules. In addition to the work by Ross, the use of constraints on sets of facts to enhance R-acyclicity and R-stratification resembles the use of dependencies, called *extensional constraints*, on ABoxes for DL query rewriting optimisation [209, 206]. Finally constraints can be used in a similar way to generalise MFA and MSA (which were introduced in Chapter 4), because for a program that satisfies a set of constraints checking universal MFA and MSA would be sufficient using a 'relaxed' critical instance.

Chapter 7

Knowledge Base Design

In this chapter we provide a high-level description of the knowledge bases (KBs), i.e. the set of nonmonotonic existential rules, that we used for our empirical evaluation. Due to lack of available test data, we assembled our own KBs and used them to conduct experiments. The rules of these KBs mainly capture knowledge of chemical nature and the experiments consist in structure-based classification of molecular entities. In this chapter we introduce the shape of the rules that form part of the constructed KBs using examples.

In order to facilitate the creation of ontological descriptions by application experts not familiar with logical formulas, we designed a 'less logician-like' syntax for nonmonotonic existential rules—the DGLP syntax. Axioms constructed using DGLP syntax are uniquely translatable to nonmonotonic existential rules. In spite of the fact that we do not provide automated support for converting DGLP axioms into rules, we provide the DGLP descriptions for a number of examples to illustrate the style of the syntax. The technical specification of the DGLP syntax can be found in Appendix A.

Section 7.1 introduces the DGLP syntax and discusses a potential extension of it with the SMILES chemical notation. Section 7.2 discusses the implicit hydrogens assumption and the impact that it has on the construction of our KB. Sections 7.3, 7.4 and 7.5 describe the types of rules that form part of the KBs; these rules correspond to representation of molecules, chemical classes and functional groups, respectively.

Finally, Section 7.6 presents an example of subsumption computation.

7.1 DGLP Syntax

As we pointed out in Section 3.3.3, introducing a notation for the representation of structures that is closer to natural language than to mathematical logic could assist ontology developers in the design of knowledge bases. Such a notation could enable knowledge modellers to produce succinct statements that are less verbose than natural language sentences but still avoid the use of special symbols. For instance, a core feature of our syntax is the use of shortcuts for representing labelled graphs, which usually have a complicated mathematical definition. To this end, we present a 'surface syntax' for nonmonotonic existential rules—that is a syntax designed to facilitate the creation of ontological descriptions by users not familiar with logic programming.

We specify this syntax by means of a BNF grammar. We refer to a **DGLP** expression, which is defined recursively by the rules of tables A.1, A.2 and A.3 (given in Section A.1 of the appendix), as a *DGLP (Description Graph Logic Programs) axiom* and to a set of DGLP axioms as a *DGLP ontology*. The union of the sets of BNF rules contained in these tables is the *DGLP syntax*. DGLP syntax is largely inspired by the OWL Manchester syntax and the syntax suggested for OWL 2 rules [123, 94]. Our intention is to maximise the overlap between the syntaxes suggested for OWL and the syntax presented here in order to exploit previous experience of knowledge engineers with these notations.

In order to support ontology developers with reasoning services, we additionally need to define a mapping from DGLP axioms to nonmonotonic existential rules. Tables A.4–A.13 (given in Section A.2 of the appendix) present the transformation operator NER that maps a set of DGLP axioms as defined from Tables A.1–A.3 to a set of nonmonotonic existential rules. The style of the definitions listed in Tables A.4–A.13 is influenced by the W3C recommendation that specifies the mapping from OWL axioms to RDF graphs [101].

To define objects with complex internal structure, DGLP axioms use the inter-

mediate definition of a *description graph*, i.e. a construct that encodes a non-empty labelled graph with the additional restriction that edges must have non-empty labels. For example the molecular structure of housane (Figure 7.1 on the left) can be represented with the following DGLP axiom, where the expression Graph(...) represents a description graph. Please note that in the following DGLP axiom, we have inserted some additional whitespaces and newlines to enhance readability; in the remainder of this chapter, we take the liberty to add or remove whitespaces and newlines where appropriate for the sake of clarity.

```
housane SubClassOf
molecule  AND
hasAtom  SOME Graph(Nodes (1 carbon, 2 carbon, 3 carbon, 4 carbon, 5 carbon)
                    Edges (1 2 singleBond, 2 3 singleBond, 3 4 singleBond,
                           4 5 singleBond, 5 1 singleBond, 2 5 singleBond)
```

Using the mappings from DGLP axioms to rules, the above DGLP axiom can be translated to the following rule.

$$\mathsf{housane}(x) \rightarrow \exists_{i=1}^{5} y_i.\ \bigwedge_{i=1}^{5} \mathsf{hasAtom}(x, y_i) \wedge \bigwedge_{i=1}^{5} \mathsf{carbon}(y_i) \wedge \bigwedge_{i=1}^{5} \mathsf{singleBond}(y_i, y_{i \bmod 5}) \wedge \mathsf{singleBond}(y_2, y_5)$$

In spite of the fact that DGLP syntax restricts the shape of the rules, it does not offer any acyclicity or stratification guarantees for the constructed knowledge base. As a consequence, after transforming a DGLP ontology into rules one needs to perform an acyclicity and a stratification check in order to ensure finiteness and uniqueness of the stable model. It would be an interesting subject for future work to design and implement support that would guide users to convert a cyclic DGLP ontology into an acyclic one (if that is possible).

7.1.1 DGLP/SMILES Notation for Chemistry

We next suggest how DGLP syntax could be further extended with SMILES,[1] a compact, widely used chemical notation for the representation of molecular graphs.

[1] SMILES stands for Simplified Molecular-Input Line-Entry System

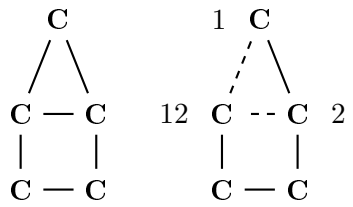


Figure 7.1: Housane structure (left) and SMILES cycle breaking (right)

SMILES was developed in the late 1980s as a specification for the description of chemical structures with short ASCII strings [238, 240, 239]. Reproducing the full SMILES specification is beyond the scope of this thesis. What we provide next is a brief summary of the key SMILES features and how the SMILES notation could be incorporated to the DGLP syntax in the future.

In a SMILES string, atoms are denoted with the IUPAC chemical element abbreviation and bonds with the symbols $-$, $=$, $\#$ and : for single, double, triple and aromatic bonds, respectively; if no bond is specified, a single bond is assumed by default. Moreover, the SMARTS notation,[2] which is a query language for SMILES strings, permits the use of wildcard atoms and bonds with the symbols $*$ and $\sim$, respectively. The SMILES string of a chemical graph is obtained by concatenating the atom symbols encountered in a depth-first tree traversal of the graph. If the chemical graph contains rings, the rings are broken to turn the graph into its spanning tree; in that case, the symbols of the atoms, whose bond has been removed to break a cycle, are extended with numeric suffix labels. During the traversal, parentheses are used to enclose branches of the tree that hang off the main backbone.

Figure 7.1 on the right shows the cycle breaks for the two rings of housane that suggest a depth-first tree traversal of the remaining tree of the graph; the broken bonds are indicated with a dashed line. A possible SMILES string for housane is C1C2CCC12, where the last carbon has both numerical suffixes '1' and '2'.

DGLP syntax can be extended with SMILES strings, by replacing the rule that

[2]SMARTS stands for SMiles ARbitrary Target Specification [3].

defines **DescriptionGraph** of Table A.1 with

DescriptionGraph ::= 'Graph' '(' **NodeSet** **EdgeSet** ')'
| 'Smiles(' **SmilesName** ')'

where **SmilesName** is a valid SMILES string enhanced with the additional SMARTS features that allow for atoms and bonds of any kind. A DGLP axiom can then be mapped to a nonmonotonic existential rule by first converting the SMILES string into a description graph and then applying the NER mapping as defined in Table A.4. For the conversion into a description graph, translation into a labelled graph representation would suffice; this could be performed by one of the many off-the-shelf conversion software tools[3] that implement an algorithm designed for this purpose (known as Structure Diagram Generation [114]) and subsequently convert the names of atoms and bonds into strings starting with a lowercase letter. For example, a DGLP axiom extended with SMILES for housane could be the following:

housane SubClassOf

molecule AND hasAtom SOME Smiles(C1C2CCC12)

Finally, an advantage of describing molecules with DGLP axioms (extended with SMILES) instead of SMILES strings is that multiple SMILES strings might exist for the same molecule, whereas DGLP axioms can be translated into formulae that are logically equivalent.

7.2 Implicit Hydrogens Assumption

As we will see in Chapter 8, we carried out different kinds of experiments in some of which we adopted the *implicit hydrogens assumption* and in some of which we did not. The implicit hydrogens assumption is an optimisation frequently encountered in computational chemistry applications according to which hydrogens of molecules are not represented explicitly but are maintained as an implicit hydrogen count associated with each atom. Since the number of hydrogens in a molecule is usually half of the

[3] A list of downloadable and web-based conversion utilities can be found at `http://www.opensmiles.org/opensmiles.html`

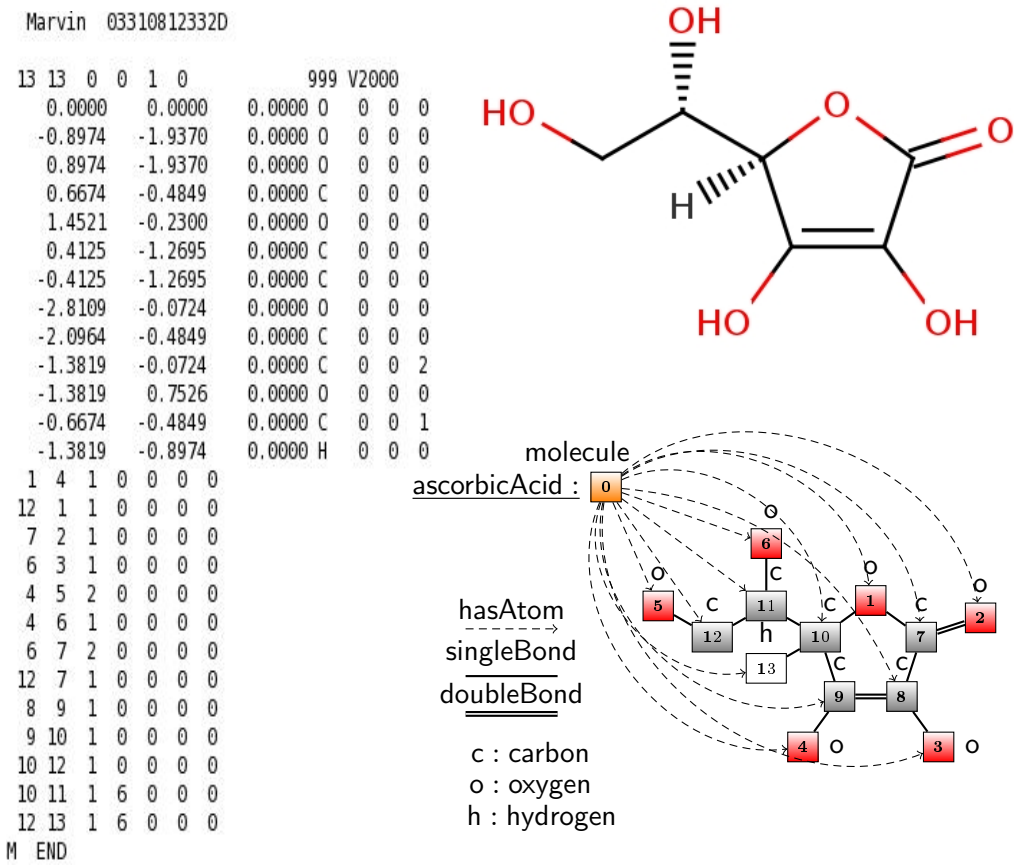


Figure 7.2: Molfile (left), chemical graph (top right) and labelled graph abstraction (bottom right) encoding the molecular structure of ascorbic acid.

total number of atoms, a significant performance improvement can be gained due to reduced memory consumption. However, as we discuss in Section 7.5 in certain cases we have to drop the implicit hydrogens assumption as it compromises R-stratification of the KB. In the description of the rules that follows we state clearly how the implicit or explicit representation of hydrogens influences the construction of the rules.

7.3 Representation of Molecules

Next, we describe how molecules can be represented by rules; we use as an example the molecule of ascorbic acid (commonly known as vitamin C). The chemical graph of ascorbic acid is depicted in the top right corner of Figure 7.2; the numerical table in

the left column of the same figure is the *molfile* of ascorbic acid, one of the standard chemical file formats (molfiles specify the atoms, the coordinates of the atoms and the bonds of a molecule using a well-defined table format). Conceptually, the structure of ascorbic acid can be abstracted with the help of a labelled graph such as the one that appears in the bottom right of Figure 7.2. In order to simplify the presentation of that graph a legend is used for the edge labels and the node labels are abbreviated; all arrowless edges are assumed to be bidirectional. For our chemical setting, the description graph of a molecule is a labelled graph whose nodes correspond to the atoms of the molecule (nodes 1–13 for ascorbic acid) plus an extra node for the molecule itself (node 0) and whose edges correspond to the bonds of the molecule (e.g. (1,7)) plus some additional edges that connect the molecule node with each one of the atom nodes (e.g. (0,1)); additionally, the atom nodes are labelled with the respective chemical elements (e.g. oxygen for node 1) and the bond edges with the corresponding bond order (e.g. singleBond for (1,7)); finally, the molecule node is labelled with molecule and the edges that connect the molecule node with each of the atom nodes are labelled with hasAtom. Both the nodes and the edges can have multiple labels allowing us to also capture molecular properties such as charge values for atoms.

Ascorbic acid can be described by the following DGLP axiom.

```
ascorbicAcid SubClassOf
molecule  AND
hasAtom  SOME Graph (Nodes (1 oxygen, 2 oxygen, 3 oxygen, 4 oxygen,
                            5 oxygen, 6 oxygen, 7 carbon, 8 carbon,
                            9 carbon, 10 carbon, 11 carbon, 12 carbon,
                            13 hydrogen)
                     Edges (1 2 singleBond, 1 10 singleBond, 2 7 doubleBond,
                            3 8 singleBond, 4 9 singleBond, 5 12 singleBond,
                            6 11 singleBond, 7 1 singleBond, 8 7 singleBond,
                            9 8 doubleBond, 10 9 singleBond, 11 10 singleBond
                            12 11 singleBond, 13 10 singleBond ) )
```

The following is the result of converting the above DGLP axiom into a rule and then skolemising it; in fact in our implementation we need a separate rule for each conjunct in the head but we use only one rule here to simplify the presentation (since the splitting into several rules happens after skolemisation, it does not affect the computed stable models). In the description that follows, we include only one direction of the bonds in order to simplify the presentation.

$$\begin{aligned}\mathsf{ascorbicAcid}(x) \rightarrow\ & \mathsf{molecule}(x) \wedge \bigwedge_{i=1}^{13} \mathsf{hasAtom}(x, f_i(x)) \wedge \bigwedge_{i=1}^{6} \mathsf{oxygen}(f_i(x)) \wedge \\ & \bigwedge_{i=7}^{12} \mathsf{carbon}(f_i(x)) \wedge \mathsf{hydrogen}(f_{13}(x)) \wedge \mathsf{singleBond}(f_8(x), f_3(x)) \wedge \\ & \mathsf{singleBond}(f_9(x), f_4(x)) \wedge \bigwedge_{i=1,9,11,13} \mathsf{singleBond}(f_{10}(x), f_i(x)) \wedge \\ & \bigwedge_{i=5,11} \mathsf{singleBond}(f_{12}(x), f_i(x)) \wedge \bigwedge_{i=1,8} \mathsf{singleBond}(f_7(x), f_i(x)) \wedge \\ & \mathsf{singleBond}(f_{11}(x), f_6(x)) \wedge \mathsf{doubleBond}(f_2(x), f_7(x)) \wedge \\ & \mathsf{doubleBond}(f_8(x), f_9(x))\end{aligned}$$

Depending on whether we adopt the implicit hydrogen assumption or not, we have additional hydrogen nodes in the graph and, thus, in the corresponding rule (the actual ascorbic acid molecule contains eight hydrogens in total). In the ascorbic acid descriptions of Figure 7.2, hydrogens are implicit (the hydrogen in node 13 is an exception because it participates in a stereochemical bond).

7.4 Representation of Chemical Classes

Before presenting the modelling of various chemical classes, we demonstrate how we can encode background chemical knowledge with DGLP axioms that can subsequently be mapped to rules. Three such axioms appear next.

bond SuperPropertyOf

singleBond OR doubleBond OR tripleBond

charged SuperClassOf

positive OR negative

horc SuperClassOf

hydrogen OR carbon

Examples of such knowledge include the fact that single and double bonds are kinds of bonds and that atoms with positive or negative charge are charged; we can also denote a particular class of atoms, e.g. atoms that are hydrogens or carbons. The translation of the abovementioned DGLP axioms into rules appears below.

$$\mathsf{singleBond}(x_1, x_2) \rightarrow \mathsf{bond}(x_1, x_2)$$

$$\mathsf{doubleBond}(x_1, x_2) \rightarrow \mathsf{bond}(x_1, x_2)$$

$$\mathsf{tripleBond}(x_1, x_2) \rightarrow \mathsf{bond}(x_1, x_2)$$

$$\mathsf{negative}(x) \rightarrow \mathsf{charged}(x)$$

$$\mathsf{positive}(x) \rightarrow \mathsf{charged}(x)$$

$$\mathsf{hydrogen}(x) \rightarrow \mathsf{horc}(x)$$

$$\mathsf{carbon}(x) \rightarrow \mathsf{horc}(x)$$

Next we describe representative examples of the classes we captured using rules; our chemical modelling was based on the textual definitions found in the ChEBI ontology.[4] The classes that we covered can be categorised into four groups depending on the feature that is used for their definition: existence of subcomponents, exact cardinality of parts, exclusive composition and cyclicity. Here we show in full detail only some of the rules; the complete logic programs are available online [2].

7.4.1 Existence of Subcomponents

The great majority of the existing chemical classes is defined via containment of atoms, functional groups or other atom arrangements. Next we show surface syntax axioms that define classes which depend on contained parts such as carbon molecular entities, polyatomic entities, carboxylic acids, esters, molecules that contain a four-membered ring and heteroorganic entities. In the following axioms we use the keyword 'GraphNL' in contrast to the previously used 'Graph' as the DGLP syntax

[4] `http://www.ebi.ac.uk/chebi/`

grammar requires the use of the former when specifying nodes that are either labelled with negative literals or are specified to be disjoint. Please note that the use of the keyword SOME in the axioms below is different than in the previous axioms as it originates from an existential quantification enclosed in the body of a rule, which is in fact a universal quantification. Also, we choose to use the additional keywords SuperClassOf and SuperPropertyOf instead of reversing the order of expressions, as it is more intuitive to state upfront the entity to be defined or recognised. Textual explanations of the DGLP axioms can be found after them.

```
carbonEntity SuperClassOf
hasAtom SOME carbon

polyatomicEntity SuperClassOf
molecule AND hasAtom SOME GraphNL (DisjointNodes ( 1, 2)
                                   Edges ( ) )

middleOxygen SuperClassOf
oxygen AND bond SOME GraphNL (DisjointNodes ( 1, 2)
                              Edges ( ) )

carboxylicAcid SuperClassOf
hasAtom SOME GraphNL (Nodes ( 1 carbon, 2 oxygen, 3 oxygen
                              NOT middleOxygen NOT charged,
                             4 horc)
                      Edges ( 1 2 doubleBond, 1 3 singleBond,
                              1 4 singleBond ) )

carboxylicEster SuperClassOf
hasAtom SOME Graph (Nodes ( 1 carbon, 2 oxygen,
                            3 oxygen, 4 carbon, 5 horc)
                    Edges ( 1 2 doubleBond, 1 3 singleBond,
                            1 5 singleBond, 3 4 singleBond ) )
```

fourMemberedRingEntity SuperClassOf

hasAtom SOME GraphNL(DisjointNodes (1 , 2 , 3 , 4)

Edges (1 2 bond, 2 3 bond, 3 4 bond, 4 1 bond))

heteroOrganicEntity SuperClassOf

hasAtom SOME GraphNL(Nodes (1 carbon, 2 NOT carbon NOT hydrogen)

Edges (1 2 bond))

We define as carbon molecular entities the molecules that contain carbon; polyatomic entities are the entities that contain at least two different atoms. Carboxylic acids are defined as molecules containing at least one carboxy group (carboxy group is a functional group with formula C(=O)OH, see Figure 7.3) attached to a carbon or hydrogen; in the representation described here we follow the implicit hydrogens assumption and thus we are not able to distinguish between an oxygen and a hydroxy group and, so, we need to specify that the oxygen of the hydroxy group is not charged (NOT charged) and participates in only one bond (NOT middleOxygen). Similarly, carboxylic esters contain a carbonyl group connected to an oxygen ((C=O)O) which is further attached to two atoms that are carbon or hydrogen. Finally, heteroorganic entities are entities which contain a carbon connected through a bond to an atom that is neither carbon nor hydrogen. One can find below the corresponding translations into rules.

$$\mathsf{molecule}(x) \wedge \mathsf{hasAtom}(x,z) \wedge \mathsf{carbon}(z) \rightarrow \mathsf{carbonEntity}(x)$$

$$\mathsf{molecule}(x) \wedge \mathsf{hasAtom}(x,z_1) \wedge \mathsf{hasAtom}(x,z_2) \wedge z_1 \neq z_2 \rightarrow \mathsf{polyatomicEntity}(x)$$

$$\mathsf{oxygen}(x) \wedge \bigwedge_{i=1}^{2} \mathsf{bond}(x,z_i) \wedge z_1 \neq z_2 \rightarrow \mathsf{middleOxygen}(x)$$

$$\bigwedge_{i=1}^{4} \mathsf{hasAtom}(x,z_i) \wedge \mathsf{carbon}(z_1) \wedge$$

$$\mathsf{oxygen}(z_2) \wedge \mathsf{oxygen}(z_3) \wedge \mathsf{horc}(z_4) \wedge$$

$$\mathsf{double}(z_1,z_2) \wedge \mathsf{single}(z_1,z_3) \wedge \mathsf{single}(z_1,z_4) \wedge$$

$$\mathbf{not}\ \mathsf{middleOxygen}(z_3) \wedge\ \mathbf{not}\ \mathsf{charged}(z_3) \rightarrow \mathsf{carboxylicAcid}(x)$$

$$\bigwedge_{i=1}^{5} \mathsf{hasAtom}(x, z_i) \bigwedge_{i=1,4} \mathsf{carbon}(z_i) \wedge$$

$$\bigwedge_{i=2,3} \mathsf{oxygen}(z_i) \wedge \mathsf{horc}(z_5) \wedge \mathsf{double}(z_1, z_2) \wedge$$

$$\bigwedge_{i=3,5} \mathsf{single}(z_1, z_i) \wedge \mathsf{single}(z_3, z_4) \rightarrow \mathsf{carboxylicEster}(x)$$

$$\bigwedge_{i=1}^{4} \mathsf{hasAtom}(x, z_i) \wedge \bigwedge_{i=1}^{4} \mathsf{bond}(z_i, z_{(i \mod 4)+1}) \wedge$$

$$\bigwedge_{1 \leq i < j \leq 4} z_i \neq z_j \rightarrow \mathsf{fourMemberedRingEntity}(x)$$

$$\bigwedge_{i=1}^{2} \mathsf{hasAtom}(x, z_i) \wedge \mathsf{carbon}(z_1) \wedge \ \mathbf{not}\ \mathsf{carbon}(z_2) \wedge$$

$$\mathbf{not}\ \mathsf{hydrogen}(z_2) \wedge \mathsf{bond}(z_1, z_2) \rightarrow \mathsf{heteroOrganicEntity}(x)$$

Apart from the examples of chemical classes given above, we are also able to recognise compounds that satisfy a disjunctive condition. E.g. in order to recognise molecules that contain one functional group or another, we can add two rules with the same head and different bodies. However, the inverse—that is classes defined with a disjunction—would not be possible because it would lead to multiple stable models.

Besides class recognition using atom containment, we can recognise classes depending on the containment of functional groups. To achieve that, we would need rules that identify the existence of functional groups, e.g. for the case of hydroxy group:

$$\mathsf{hasAtom}(x, z_1) \wedge \mathsf{oxygen}(z_1)$$

$$\mathsf{hasAtom}(x, z_2) \wedge \mathsf{hydrogen}(z_2)$$

$$\mathsf{singleBond}(z_1, z_2) \rightarrow \exists y.\mathsf{hasFunctionalGroup}(x, y) \wedge \mathsf{hydroxyGroup}(y)$$

Nevertheless, this could lead to incorrect modelling as it could derive multiple terms for groups with symmetric structure. One can amend that using rules with equality atoms in the head, but since this is not covered by our formal framework, we omitted this approach in the design of our knowledge base.

7.4.2 Exact Cardinality of Parts

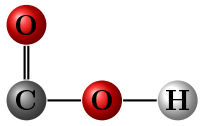


Figure 7.3: Chemical structure of carboxy group

Chemical classes can also be defined by specifying the exact number of contained atoms or functional groups, similarly to qualified number restrictions in Description Logics [13]. Examples include molecules that contain only one atom, molecules with exactly two carbons and dicarboxylic acids, that is molecules with exactly two carboxy groups (see Figure 7.3 for the structure of carboxy groups). DGLP axioms for the definition of molecules with exactly two carbons and for the definition of dicarboxylic acids appear next.

```
twoCarbonMolecule SuperClassOf
molecule  AND hasAtom  EXACTLY 2 carbon

diCarboxylicAcid SuperClassOf
molecule  AND
hasAtom   EXACTLY 2 GraphNL(Nodes (1 carbon, 2 oxygen, 3 oxygen,
                                   4 hydrogen)
                            Edges ( 1 2 doubleBond, 1 3 singleBond,
                                    3 4 singleBond ) )
```

The translation into rules follows. One can readily verify that the DGLP syntax formulation is more direct and intuitive than its equivalent translation into rules.

$$\bigwedge_{i=1}^{3} \mathsf{hasAtom}(x, z_i) \wedge \bigwedge_{i=1}^{3} \mathsf{carbon}(z_i) \wedge$$
$$z_1 \neq z_2 \wedge z_2 \neq z_3 \wedge z_1 \neq z_3 \rightarrow \mathsf{atLeastThreeCarbon}(x)$$

$$\mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{2} \mathsf{hasAtom}(\mathsf{x}, \mathsf{z_i}) \wedge \bigwedge_{\mathsf{i}=1}^{2} \mathsf{carbon}(\mathsf{z_i}) \wedge$$

$$\wedge \mathsf{z}_1 \neq \mathsf{z}_2 \wedge \ \mathbf{not}\ \mathsf{atLeastThreeCarbon}(\mathsf{x}) \rightarrow \mathsf{twoCarbonMolecule}(\mathsf{x})$$

$$\bigwedge_{\mathsf{i}=1}^{12} \mathsf{hasAtom}(\mathsf{x}, \mathsf{z_i}) \wedge \bigwedge_{\mathsf{i}=1,5,9} \mathsf{carbon}(\mathsf{z_i}) \wedge$$

$$\bigwedge_{\mathsf{i}=2,3,6,7,10,11} \mathsf{oxygen}(\mathsf{z_i}) \wedge \bigwedge_{\mathsf{i}=1,4,8} \mathsf{singleBond}(\mathsf{z_i}, \mathsf{z_{i+2}}) \wedge$$

$$\bigwedge_{\mathsf{i}=3,7,11} \mathsf{singleBond}(\mathsf{z_i}, \mathsf{z_{i+1}}) \wedge \bigwedge_{\mathsf{i}=4,8,12} \mathsf{hydrogen}(\mathsf{z_i}) \wedge$$

$$\bigwedge_{\mathsf{i}=1,4,8} \mathsf{doubleBond}(\mathsf{z_i}, \mathsf{z_{i+1}}) \wedge \bigwedge_{1 \leq \mathsf{i} < \mathsf{j} \leq 12} \mathsf{z_i} \neq \mathsf{z_j} \rightarrow \mathsf{atLeastThreeCarboxy}(\mathsf{x})$$

$$\mathsf{molecule}(\mathsf{x}) \wedge \bigwedge_{\mathsf{i}=1}^{8} \mathsf{hasAtom}(\mathsf{x}, \mathsf{z_i}) \wedge$$

$$\bigwedge_{\mathsf{i}=1,5} \mathsf{carbon}(\mathsf{z_i}) \wedge \bigwedge_{\mathsf{i}=2,3,6,7} \mathsf{oxygen}(\mathsf{z_i}) \wedge$$

$$\bigwedge_{\mathsf{i}=4,8} \mathsf{hydrogen}(\mathsf{z_i}) \wedge \bigwedge_{\mathsf{i}=1,4} \mathsf{doubleBond}(\mathsf{z_i}, \mathsf{z_{i+1}}) \wedge$$

$$\bigwedge_{\mathsf{i}=1,4} \mathsf{singleBond}(\mathsf{z_i}, \mathsf{z_{i+2}}) \wedge \bigwedge_{\mathsf{i}=3,7} \mathsf{singleBond}(\mathsf{z_i}, \mathsf{z_{i+1}}) \wedge$$

$$\bigwedge_{1 \leq \mathsf{i} < \mathsf{j} \leq 8} \mathsf{z_i} \neq \mathsf{z_j} \wedge \ \mathbf{not}\ \mathsf{atLeastThreeCarboxy}(\mathsf{x}) \rightarrow \mathsf{diCarboxylicAcid}(\mathsf{x})$$

7.4.3 Exclusive Composition

We next present classes of molecules such that each atom (or bond) they contain satisfies a particular property. These features are usually very naturally modelled with the help of nonmonotonic negation. Examples include inorganic molecules that consist exclusively of non-carbon atoms. In spite of the fact that there are many compounds with carbons considered inorganic, in this thesis we align our encoding with the ChEBI definition of inorganic molecular entities (ChEBI id 24835), according to which no carbons occur in these entities; however, if the modeller wishes it, it is straightforward to declare exceptions with the help of nonmonotonic negation. Another example is the class of hydrocarbons which only contain hydrogens and

carbons; also saturated compounds are defined as the compounds whose carbon to carbon bonds are all single. The corresponding DGLP axioms appear next.

```
inorganic SuperClassOf
molecule AND hasAtom ONLY ( NOT carbon)

hydroCarbon SuperClassOf
carbonEntity AND hasAtom ONLY (hydrogen OR carbon)

unsaturated SuperClassOf
molecule AND hasAtom SOME Graph (Nodes ( 1 carbon, 2 carbon)
                                 Edges ( 1 2 doubleBond ) )

unsaturated SuperClassOf
molecule AND hasAtom SOME Graph (Nodes ( 1 carbon, 2 carbon)
                                 Edges ( 1 2 tripleBond ) )

saturated SuperClassOf
molecule AND NOT unsaturated
```

The DGLP axiom for hydroCarbon is translated into two rules, where an auxiliary predicate is used, as specified by Table A.8. Please note that one can use more than one DGLP axioms (and thus rules) to define classes that emerge as a result of different structural configurations, which is the case for unsaturated molecules. Below we list the respective translation into rules.

$$\text{molecule}(x) \wedge \textbf{not}\ \text{carbonEntity}(x) \rightarrow \text{inorganic}(x)$$

$$\text{hasAtom}(x, z) \wedge \textbf{not}\ \text{carbon}(z) \wedge \textbf{not}\ \text{hydrogen}(z) \rightarrow \text{AuxhydroCarbon}(x)$$

$$\text{carbonEntity}(x) \wedge \textbf{not}\ \text{AuxhydroCarbon}(x) \rightarrow \text{hydroCarbon}(x)$$

$$\text{molecule}(x) \wedge \text{hasAtom}(x, z_1) \wedge \text{carbon}(z_1)$$
$$\text{hasAtom}(x, z_2) \wedge \text{carbon}(z_2) \wedge \text{doubleBond}(z_1, z_2) \rightarrow \text{unsaturated}(x)$$

$$\text{molecule}(x) \wedge \text{hasAtom}(x, z_1) \wedge \text{carbon}(z_1)$$
$$\text{hasAtom}(x, z_2) \wedge \text{carbon}(z_2) \wedge \text{tripleBond}(z_1, z_2) \rightarrow \text{unsaturated}(x)$$

$$\mathsf{molecule(x)} \wedge \ \mathbf{not}\ \mathsf{unsaturated(x)} \rightarrow \mathsf{saturated(x)}$$

7.4.4 Cyclicity-Related Classes

These chemical classes include the definition of molecules containing a ring of any length as well as other definitions that depend on the cyclicity of molecules, such as alkanes. We define cyclic molecules as molecules containing a loop of length at least three. To this end, one first needs to identify an atom that participates in such a loop (or one of its neighbours) and subsequently flag the molecule that contains it as cyclic. The following rules encode this effect:

$$\mathsf{bond(x,y)} \wedge \mathsf{bond(y,z)} \wedge \mathsf{x} \neq \mathsf{y} \wedge \mathsf{y} \neq \mathsf{z} \wedge \mathsf{x} \neq \mathsf{z} \rightarrow \mathsf{path3DistNodes(x,y,z)} \quad (r_1)$$

$$\mathsf{path3DistinctNodes(x,y,z)} \wedge \mathsf{bond(z,w)} \wedge \mathsf{x} = \mathsf{w} \wedge \mathsf{y} \neq \mathsf{w} \wedge \mathsf{z} \neq \mathsf{w} \rightarrow \mathsf{path3DistNodes(x,z,w)} \quad (r_2)$$

$$\mathsf{path3DistNodes(x,y,z)} \wedge \mathsf{bond(z,x)} \rightarrow \mathsf{loopAtLeast3Atoms(x)} \quad (r_3)$$

$$\mathsf{molecule(x)} \wedge \mathsf{hasAtom(x,y)} \wedge \mathsf{loopAtLeast3Atoms(y)} \rightarrow \mathsf{cyclic(x)} \quad (r_4)$$

DGLP syntax allows for arbitrarily shaped nonmonotonic existential rules that do not need to be transformed as it is indicated by the definition of the **Rule** expression in Table A.1. This is useful for the rules r_1–r_4 which can be expressed in the DGLP syntax by enclosing them into parentheses and adding the NERule prefix. Please note that rules r_1–r_4 recognise molecules that contain cycles but cannot be used to enforce containment of cyclic parts by cyclic molecules. Also, rules r_1–r_4 may recognise entities with cycles of unbounded length or entities that satisfy a disjunction of conditions (e.g. containing a cycle of length three four), but cannot be used to define such entities. Once cyclic molecules are defined one can exploit nonmonotonic negation to recognise other classes such as alkanes, which are saturated, non-cyclic hydrocarbons. The DGLP axiom for alkanes and its rule translation appear next.

alkane SuperClassOf

saturated AND hydroCarbon AND NOT cyclic

$$\mathsf{saturated(x)} \wedge \mathsf{hydroCarbon(x)} \wedge \ \mathbf{not}\ \mathsf{cyclic(x)} \rightarrow \mathsf{alkane(x)}$$

7.5 Representation of Functional Groups

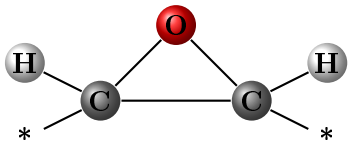


Figure 7.4: Epoxy group structure

Chemical entities containing functional groups are structured objects that can act both as superclasses and subclasses of other entities. For example, epoxide, which is an entity containing epoxy (epoxy's chemical structure appears in Figure 7.4), can be a superclass of molecules that contain the epoxy group and at the same time a subclass of more generic classes such as carbon-containing entities. Epoxide can be represented with the following DGLP axiom.

epoxide EquivalentTo

hasAtom SOME Graph(Nodes (1 oxygen, 2 carbon, 3 carbon, 4 hydrogen,
5, 6, 7 hydrogen)
Edges (1 2 singleBond, 2 3 singleBond, 3 1 singleBond,
2 4 singleBond, 2 5 singleBond, 3 6 singleBond,
3 7 singleBond))

The translation of the above axiom into rules appears next.

$$\bigwedge_{i=1}^{7} \mathsf{hasAtom}(x, z_i) \wedge \mathsf{oxygen}(z_1) \wedge \bigwedge_{i=2,3} \mathsf{carbon}(z_i) \wedge$$
$$\bigwedge_{i=4,7} \mathsf{hydrogen}(z_i) \wedge \bigwedge_{i=1}^{3} \mathsf{singleBond}(z_i, z_{(i \mod 3)+1}) \wedge$$
$$\bigwedge_{i=4,5} \mathsf{singleBond}(z_2, z_i) \wedge \bigwedge_{i=6,7} \mathsf{singleBond}(z_3, z_i)) \wedge$$
$$\bigwedge_{i=1}^{7} \textbf{not } \mathsf{newEpoxide}(z_i) \rightarrow \mathsf{epoxide}(x) \wedge \mathsf{recEpoxide}(x)$$

$$\mathsf{epoxide(x)} \wedge \ \mathbf{not}\ \mathsf{recEpoxide(x)} \rightarrow \exists_{i=1}^{7} \mathsf{y_i}.\bigwedge_{i=1}^{7} \mathsf{hasAtom(x, y_i)} \wedge \mathsf{oxygen}(y_1) \wedge \bigwedge_{i=2,3} \mathsf{carbon(y_i)} \wedge \bigwedge_{i=4,7} \mathsf{hydrogen(y_i)} \wedge \bigwedge_{i=1}^{3} \mathsf{singleBond}(y_i, y_{(i \mod 3)+1}) \wedge \bigwedge_{i=4,5} \mathsf{singleBond}(y_2, y_i) \wedge \bigwedge_{i=6,7} \mathsf{singleBond}(y_3, y_i) \wedge \bigwedge_{i=1}^{7} \mathsf{newEpoxide(y_i)}$$

Similarly, we can represent a variety of classes depending on functional groups such as phosphate, disulphide and hydrazone. Armengol and Plaza [10] provide a useful resource for frequently encountered functional groups. For this setting we do not operate on the implicit hydrogens assumption, because this would result in a violation of R-stratification as is indicated by the following example, which involves the modelling of carboxylic acid.

$$\mathsf{oxygen(x)} \wedge \bigwedge_{i=1}^{2} \mathsf{bond(x, y_i)} \wedge y_1 \neq y_2 \rightarrow \mathsf{middleOxygen(x)} \qquad (r_5)$$

$$\bigwedge_{i=1}^{4} \mathsf{hasAtom(x, y_i)} \wedge \mathsf{carbon}(y_1) \wedge \mathsf{oxygen}(y_2) \wedge \mathsf{oxygen}(y_3) \wedge \mathsf{horc}(y_4) \wedge \mathsf{doubleBond}(y_1, y_2) \wedge \mathsf{singleBond}(y_1, y_3) \wedge \mathsf{singleBond}(y_1, y_4) \wedge \mathbf{not}\ \mathsf{g_{ca}}(y_1) \wedge \ \mathbf{not}\ \mathsf{g_{ca}}(y_2) \wedge \ \mathbf{not}\ \mathsf{g_{ca}}(y_3) \wedge \mathbf{not}\ \mathsf{g_{ca}}(y_4) \wedge \ \mathbf{not}\ \mathsf{middleOxygen}(y_3) \wedge \mathbf{not}\ \mathsf{charged}(y_3) \rightarrow \mathsf{carboxylicAcid(x)} \wedge \mathsf{r_{ca}(x)} \qquad (r_6)$$

Input fact: $\mathsf{ascorbicAcid(a)}$
Stable model: $\mathsf{ascorbicAcid(a)}$, $\mathsf{hasAtom(a, a_i^f)}$ for $1 \leq \mathsf{i} \leq 13$, $\mathsf{oxygen(a_i^f)}$ for $1 \leq \mathsf{i} \leq 6$, $\mathsf{carbon(a_i^f)}$ for $7 \leq \mathsf{i} \leq 12$, $\mathsf{hydrogen(a_{13}^f)}$, $\mathsf{singleBond(a_8^f, a_3^f)}$, $\mathsf{singleBond(a_9^f, a_4^f)}$, $\mathsf{singleBond(a_{12}^f, a_i^f)}$ for $\mathsf{i} \in \{5, 11\}$, $\mathsf{singleBond(a_7^f, a_i^f)}$ for $\mathsf{i} \in \{1, 8\}$, $\mathsf{singleBond(a_{11}^f, a_6^f)}$, $\mathsf{doubleBond(a_2^f, a_7^f)}$, $\mathsf{bond(a_9^f, a_4^f)}$, $\mathsf{singleBond(a_{10}^f, a_i^f)}$ for $\mathsf{i} \in \{1, 9, 11, 13\}$, $\mathsf{doubleBond(a_8^f, a_9^f)}$, $\mathsf{bond(a_8^f, a_3^f)}$, $\mathsf{bond(a_{12}^f, a_i^f)}$ for $\mathsf{i} \in \{5, 11\}$, $\mathsf{bond(a_{10}^f, a_i^f)}$ for $\mathsf{i} \in \{1, 9, 11, 13\}$, $\mathsf{bond(a_7^f, a_i^f)}$ for $\mathsf{i} \in \{1, 8\}$, $\mathsf{bond(a_{11}^f, a_6^f)}$, $\mathsf{bond(a_2^f, a_7^f)}$, $\mathsf{bond(a_8^f, a_9^f)}$, $\mathsf{horc(a_i^f)}$ for $7 \leq \mathsf{i} \leq 13$, $\mathsf{molecule(a)}$, $\mathsf{organic(a)}$, $\mathsf{polyatomicEntity(a)}$, $\mathsf{cyclic(a)}$, $\mathsf{middleOxygen(a_1^f)}$, $\mathsf{carboxylicEster(a)}$, $\mathsf{unsaturated(a)}$, $\mathsf{heteroOrganicEntity(a)}$

Table 7.1: Stable model of the KB with the input fact $\mathsf{ascorbicAcid(a)}$ and the rules of Section 7.3 and 7.4; $\mathsf{f_i(a)}$ is abbreviated with $\mathsf{a_i^f}$ for $1 \leq \mathsf{i} \leq 13$.

$$\begin{aligned}\mathsf{carboxylicAcid(x)} \wedge \ \mathbf{not}\ \mathsf{r_{ca}(x)} \rightarrow \exists_{\mathsf{i}=1}^{4}\mathsf{y_i}.\ &\bigwedge_{\mathsf{i}=1}^{4}\mathsf{hasAtom(x, y_i)} \wedge \mathsf{carbon(y_1)} \wedge \\ &\mathsf{oxygen(y_2)} \wedge \mathsf{oxygen(y_3)} \wedge \mathsf{horc(y_4)} \wedge \\ &\mathsf{doubleBond(y_1, y_2)} \wedge \mathsf{singleBond(y_1, y_3)} \wedge \\ &\mathsf{singleBond(y_1, y_4)} \wedge \bigwedge_{\mathsf{i}=1}^{4}\mathsf{g_{ca}(y_i)} \qquad (r_7)\end{aligned}$$

$$\mathsf{singleBond(x, y)} \rightarrow \mathsf{bond(x, y)} \qquad (r_7)$$

We have $r_5 \dashrightarrow r_6 \dashrightarrow r_7 \xrightarrow{+} r_8 \xrightarrow{+} r_5$, that is $\{r_5, r_6, r_7, r_8\}$ is no longer R-stratified. However, if we choose to explicitly represent hydrogens, rule r_5 can be removed thus restoring R-stratification.

7.6 Determining Subclass Relations

We have already demonstrated how meaningful subsumptions can be derived by reasoning over programs under the stable model semantics. Let us revisit this process by means of an example. For the case of ascorbic acid, if we append the fact $\mathsf{ascorbicAcid(a)}$ to the KB that consists of the rules appearing in Sections 7.3 and 7.4,

we obtain the stable model that appears in Table 7.1 (for simplicity we have omitted atoms whose predicates are path3DistNodes and loopAtLeast3Atoms that are too many). From the contained unary atoms we can infer the superclasses of ascorbic acid, that is we deduce that ascorbic acid is—among others—a polyatomic, cyclic, heteroorganic molecular entity that contains carbon and a carboxylic ester. If there is no relevant atom for a chemical class in the stable model, then we conclude that class not to be a subsumer, e.g. since carboxylicAcid(a) is not found in the stable model, carboxylic acid is not a superclass of ascorbic acid.

Chapter 8

Empirical Evaluation

In this chapter we assess the practical feasibility of nonmonotonic existential rules by applying them to the automatic construction of chemical taxonomies. In particular, we test how efficiently we can compute chemical class subsumers of chemical entities using state-of-the-art datalog reasoners and the knowledge base presented in Chapter 7. To this end, we present LoPStER , a prototypical tool for ontology-based classification of chemicals, and we test its performance by carrying out a number of classification experiments.

The chapter is organised as follows. Section 8.1 introduces the problem of chemical classification and provides an overview of related work. Section 8.2 provides a high-level description of our implementation, Section 8.3 presents the experimental results of our evaluation and Section 8.4 discusses the derived subsumptions. Section 8.5 compares our work with previous logic-based approaches on chemical classification and concludes with potential applications of our framework to other domains.

8.1 The Chemical Classification Problem

Taxonomies provide a compelling way of aggregating information, as hierarchically organised knowledge is more accessible to humans. This is evidenced, e.g. by the pervasive use of the periodic table in chemistry, one of the longest-standing and most widely adopted classification schemes in natural sciences [181]. Additionally,

organising a large number of different objects into meaningful groups facilitates the discovery of significant properties pertaining to that group; these discoveries can then be used to predict features of later detected members of the group. For instance, esters with low molecular weight tend to be more volatile and, so, a newly found ester with low weight is expected to be highly volatile, too. Moreover, taxonomic structures have practical applications to life sciences, e.g. they can be exploited by bioinformaticians to produce background sets for enrichment analysis tasks [154]. As a consequence, grouping objects on the basis of shared characteristics forms an integral part of many life sciences data management systems [220]. Due to the availability of numerous performant logic-based reasoners, life scientists can employ ontological systems to drive fast, automatic and repeatable classification processes that produce high quality hierarchies [246, 57]. Therefore, in areas such as biology and chemistry with a long tradition of taxonomy use, a wealth of knowledge bases with hierarchy facilities exist and steadily grow.

A prominent example of a knowledge base for the chemical domain is the ChEBI[1] ontology, an open-access dictionary of molecular entities that provides high quality annotation and taxonomical information for chemical compounds [68, 66, 111]. ChEBI fosters interoperability between researchers by acting as the primary chemical annotation resource for various biological databases such as BioModels [156], Reactome [62] and the Gene Ontology [52]. Moreover, ChEBI supports numerous tasks of biochemical knowledge discovery such as elucidation of metabolic networks [84], analysis of metabolomic data [51] and identification of disease pathways [118].

A prerequisite for the automatic construction and maintenance of hierarchies is the availability of ontological formalisms that are expressive enough to model the entities destined for classification. As we argued in Section 3.1, state-of-the-art ontology technology falls short of expressive power when it comes to representing objects with internal structure. More precisely, the 'tree-likedness' and the open-world assumption of OWL hinder the representation of biological knowledge [189] and—as we discussed in Section 3.1—also the ability to define classes in the chemical domain. These

[1] **Ch**emical **E**ntities of **B**iological **I**nterest

inadequacies obstruct the full automation of classification for chemical ontologies, such as ChEBI, which is curated by human experts who manually annotate and determine the chemical classes of new molecular entries. Currently, ChEBI describes 32,801 fully annotated entities (release 103) and grows at a rate of approximately 4,500 entities per year (estimate based on previous releases[2]). Given the size of other publicly available chemical databases, such as PubChem [236] that contains records for 1.6 million molecules and ChemSpider [196] that encompasses more than 26 million unique molecules, there is clearly a strong potential for ChEBI to expand by speeding up the curating tasks through automation of chemical classification [132].

As the classification of compounds is a key task of the drug development process [237], the construction of chemical hierarchies has been the topic of various investigations [113] that go beyond logic-based knowledge representation (KR). In particular, prominent contributions towards automatic chemical hierarchy construction build upon statistical machine learning (ML) techniques [223, 70, 138] and algorithmic classification systems [5, 21]. As we already saw, in KR approaches molecule and class descriptions are represented with logical axioms produced by experts and subsumptions are identified with the help of automated reasoning programs; on the other hand, statistical algorithms require training through annotated datasets in order to be able to classify new entries. So, KR approaches are based on the explicit axiomatisation of knowledge, whereas ML algorithms act as a 'black box' that assigns to new entries superclasses that are likely to be correct. As a consequence, the taxonomies produced using logic-based techniques are provably correct (as long as the modelling of the domain knowledge is faithful), but the statistically produced hierarchies (although usually faster) need to be evaluated against a curated gold standard. Moreover, KR and ML techniques tend to classify chemical entities on the basis of different features: KR approaches usually exploit structural information to perform classification (e.g. saturated molecules), whereas ML techniques try to predict features from functional characteristics such as carcinogenicity [223]. Therefore, the two approaches are not directly comparable. Here, we focus on logic-based chemical clas-

[2] `http://www.ebi.ac.uk/chebi/newsForward.do`

sification [234, 144, 112, 172, 173], which in certain cases can complement statistical and algorithmic approaches [84, 110, 82, 28].

One of the first ontology-based approaches to chemical classification is reported by Villanueva-Rosales and Dumontier [234] where molecules were classified using functional group containment. Villanueva-Rosales and Dumontier manually built a hierarchy of chemical classes based on functional groups using OWL TBox axioms; subsequently, they extracted OWL ABox assertions from molecular descriptions (given in a chemical file format) and proceeded with assigning molecules (i.e. OWL individuals) to chemical classes (i.e. OWL classes) with the help of an OWL reasoning engine. However, due to the inherent inability of OWL to represent cycles, it was impossible to model classes depending on cyclic functional groups such as benzene; the authors suggested the use of DL-safe rules as a workaround, which was only partially satisfactory for reasons that we explained in Section 3.1.2.

Further to that, Konyk et al. [144] implemented a similar classification methodology, where nominal-based class expressions instead of OWL ABox assertions were used to represent molecules. However, Konyk et al. concluded that this solution had the drawback of precluding the definition of multiple instances of the same molecule. As an alternative, they suggested the adoption of the FOL Description Graphs extension, which however suffers from the deficiencies that we listed in Section 3.1.3. As a natural continuation, Hastings et al. [112] applied the FOL Description Graphs framework [184] to the representation of non-tree-shaped molecular structures. However, this allowed only for the identification of cycles with alternating single and double bonds and created a number of other modelling problems, as we explained through examples in Section 3.1.3. More recently, Dumontier outlined a modelling methodology for the inference of subsumptions depending on molecular symmetry and atomic connectivity that draws upon OWL classes [73].

In Chapters 4, 5 and 6, we laid the theoretical foundation of a new expressive ontology language that relies on nonmonotonic existential rules and is suitable for the representation of graph-shaped objects and, thus, molecules. In the remainder of this chapter, we test the nonmonotonic existential rules framework in practice

by presenting experimental results of a two-fold empirical evaluation conducted over data extracted from the ChEBI ontology. The objectives of our experiments are to assess the feasibility of logic-based techniques in order to compute (i) chemical class subsumers of molecules and (ii) chemical class subsumers of molecules and of chemical classes. These hypotheses are tested by the first and second part of our implementation, respectively.

8.2 Implementation Architecture

This section provides an overview of LoPStER,[3] the prototype we developed for structure-based chemical classification. The implementation is based on the DLV system, a powerful and efficient deductive databases and logic programming engine [153]. DLV constitutes the automated reasoning component used by LoPStER for stable model computation of a rule set. Figure 8.1 depicts the basic processing steps as well as the different files that are parsed and produced by LoPStER. LoPStER is implemented in Java and is available along with the input datasets and obtained results online [2]. Next, we describe in more detail the several stages of execution.

1. **File parsing.** LoPStER parses the molfiles [64] of the molecules to be classified using the Chemistry Development Kit Java library [225]. The molfile is a widely used chemical file format that describes molecular structures with a connection table; e.g., the molfile of ascorbic acid appears in the left of Figure 7.2. For each molecule, a labelled graph (e.g. bottom right of Figure 7.2 on page 158) representation is generated from its molfile according to the transformation outlined in Section 7.3. In this stage, text files that contain functional group descriptions (if any) are also parsed and converted into nonmonotonic existential rules as described in Section 7.5.

2. **Compilation of the KB.** For each molecule the labelled graph representation is used to produce a set of rules that encode its structure; similarly, pairs of rules are created for the functional group descriptions. These rules along with

[3] **Lo**gic **P**rogramming for **St**ructured **E**ntities **R**easoner

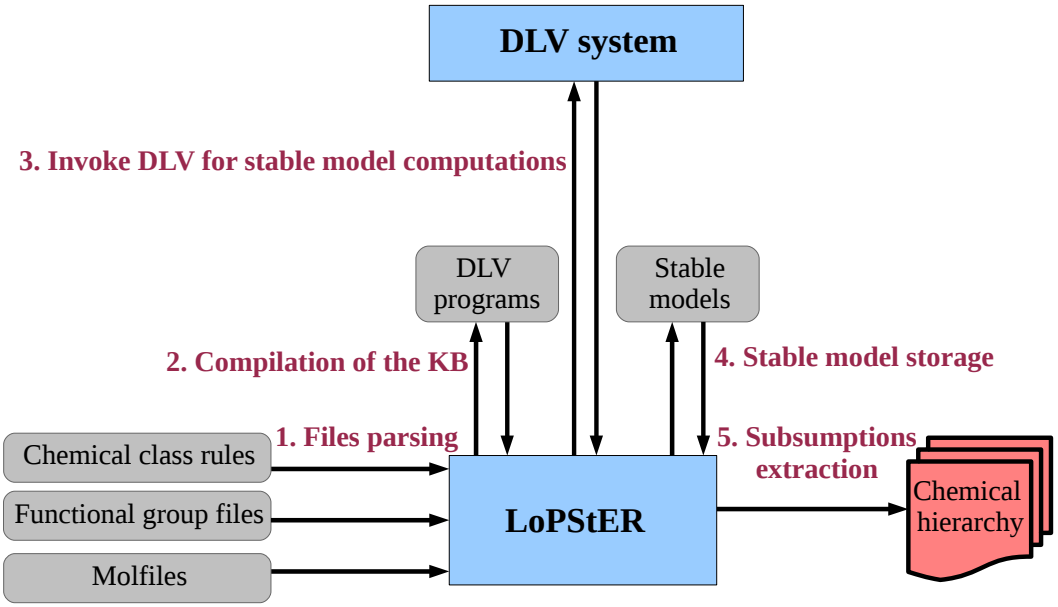


Figure 8.1: Architecture of LoPStER

the chemical class rules and the facts necessary to determine subclass relations are combined to produce DLV programs that are stored as plain text files on disk.

3. **Invoke DLV for model computation.** DLV is invoked to perform stable model computations.

4. **Stable model storage.** The stable model computed by DLV is stored in a file on disk to enable subsequent processing.

5. **Subsumptions extraction.** This is the final phase where the stable model file is parsed in order to detect the superclasses of each chemical entity. All the subsumee/subsumer pairs are stored in a separate spreadsheet file on disk.

8.3 Experimental Results

In this section, we describe the experimental results we obtained by testing our framework in practice. The conducted empirical evaluation is divided into two parts. In the first part (Section 8.3.1 [165, 169]), hydrogens are represented implicitly and we ob-

tain termination guarantees with the help of an MSA check (presented in Chapter 4); uniqueness of the stable model follows from the fact that the KB is (classically) stratified. Subsumptions obtained within this setting refer to molecule/chemical class pairs. For the second part (Section 8.3.2 [171]), hydrogens are explicitly stated; finiteness and uniqueness of the stable model is guaranteed by R-acyclicity and R-stratification, respectively. In this case, we infer subclass relations both for molecule/chemical class and chemical class/chemical class pairs. For both parts of the evaluation, the experiments were performed on a desktop computer (2GHz quadcore CPU, 4GB RAM, 2GB Java heap size) running Linux.

8.3.1 Tests with Implicit Hydrogens

In this part of the evaluation, we measured the time required by LoPStER to perform classification of molecules. To obtain test data we extracted molfile descriptions of 500 molecules from the ChEBI ontology and converted them into rules as described in Section 7.3; for each molecule, we extended the KB with a suitable fact (e.g. $\mathsf{ascorbicAcid}(\mathsf{a})$) to enable subsequent discovery of the molecules superclasses. The represented compounds were of diverse size, varying from 1 to 59 atoms. The selected 500 compounds consisted of ChEBI ids provided by Janna Hastings during a research visit at EBI and of molecules extracted from the ChEBI database using SQL queries. In the SQL queries a bound of 60 atoms per molecule was imposed due to limitations by DLV on the number of variables accepted per rule (these limitations were lifted in a later version of DLV). Moreover, we modelled chemical classes using rules such as the ones presented in Section 7.4. For these experiments no rule pairs such as the ones presented in Section 7.5 were used, that is chemical classes depending on functional groups were represented only by recognition rules (that is chemical class rules) and not by definition rules.

Since the KB that we created contains rules with function symbols in the head (such as the rule used to encode the molecular structure of ascorbic acid), we ensured termination of our reasoning process by performing an MSA check on the constructed KB. Since MSA guarantees finiteness only for positive rules, we stratified the program

(in the classical sense) and tested MSA on the lowest stratum of the KB which consists only of positive rules; let P be that set of rules. Since every existential rule of the program was in P, the MSA check was sufficient to guarantee finiteness of the stable model of the overall program. In order to perform the MSA check, we transformed P into $\mathsf{MSA}(P)$, used DLV to compute the stable model of $\mathsf{MSA}(P)$ and checked whether Cycle occurred in that model.

We investigated the scalability of our prototype by altering two different parameters of the knowledge base, namely the number of represented molecules and the type of modelled chemical classes. Initially, we constructed 10 DLV programs each of which contained rules encoding $50 \cdot i$ different compounds, where $1 \leq i \leq 10$, and rules defining chemical classes, excluding the cyclicity-related classes (48 classes in total). Next, we repeated the same construction but this time including the rules for the cyclicity-related classes (51 classes). In the rest of the section, we refer to the first setting as 'no cyclic' and to the second as 'with cyclic'. Table 8.1 depicts the number of rules obtained for each of the aforementioned DLV programs.

No of molecules	No of rules 'no cyclic'	No of rules 'with cyclic'
50	3614	3620
100	6832	6838
150	18072	18072
200	23746	23752
250	28502	28508
300	31892	31898
350	35046	35052
400	38095	38101
450	41536	41542
500	43629	43635

Table 8.1: Number of rules for programs of different size when testing with implicit hydrogens

Additionally and in order to optimise the performance, we explored how stable model computation times fluctuate depending on the size of DLV programs. In par-

No of molecules	5-mod.	10-mod.	20-mod.	25-mod.	No-mod.
50	3.998	3.444	3.188	3.084	3.400
100	3.158	2.957	2.800	2.742	3.201
150	3.579	3.315	2.939	2.946	3.768
200	4.398	4.432	3.593	3.648	4.714
250	5.682	5.111	4.935	5.085	8.061
300	7.485	6.465	5.723	5.861	8.791
350	7.446	6.416	6.392	6.619	9.471
400	8.467	8.773	8.229	7.832	11.138
450	9.955	9.531	8.770	8.588	12.655
500	10.616	9.973	9.024	9.053	12.833

Table 8.2: Classification times in seconds for programs of increasing size when split into modules of different size for 'no cyclic mode' when testing with implicit hydrogens

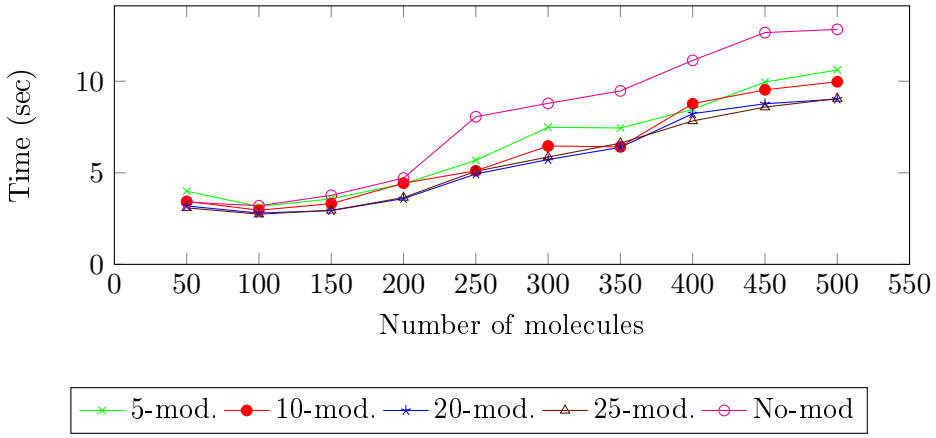


Figure 8.2: Curves of classification times for 'no cyclic' mode and testing with implicit hydrogens

ticular, we partitioned the DLV programs into modules, we measured classification times for each module separately and we summed up the times. Each module contained the facts and the rules describing a subset of the molecules represented in the initial DLV program; the rules defining chemical classes were included by each module. Thus, the size of each module depended only on the number of encoded molecules. We tested modules of size 5, 10, 20 and 25 as well as DLV programs without any partitioning for both 'no cyclic' mode and 'with cyclic' mode. We refer to each of these configurations with the names 5-mod., 10-mod., 20-mod., 25-mod. and No-mod., respectively.

All the KBs that were tested passed the MSA check. Table 8.2 summarises the classification times for the 'no cyclic' mode and for all the module-dependent experimental settings. The first column displays the number of molecules and the second to sixth columns show the time needed to perform classification where the size of the module is indicated by the column name. The time measurements of Table 8.2 count the time spent from before the molfiles parsing until after the MSA check and subsumptions extraction. Figure 8.2 illustrates the plots of the time intervals appearing in Table 8.2 with regard to the number of molecules represented by the respective DLV program. Performing classification with the modularised knowledge base tends to be faster than with the non-modularised one; we observed the shortest execution time for module sizes 20 and 25 in 'no cyclic' mode.

Table 8.3 lists analogous time measurements for 'with cyclic' mode; similarly, Figure 8.3 depicts the curves resulting from these times. Once more we observe a slight improvement in performance when splitting the program into modules. Similarly to before, the execution is optimal for module size 20. Finally, Figure 8.4 illustrates graphically the optimal classification times (module size 20) with respect to both the number of molecules and number of rules.

The abovementioned performance results are encouraging for the practical feasibility of our approach: the time measurements for 500 molecules suggest that the entire set of molecular entities that are currently represented by the ChEBI ontology (20,961 3-star entities as of release 95 [111]) could be classified in 22 minutes for

No of molecules	5-mod.	10-mod.	20-mod.	25-mod.	No-mod.
50	11.563	10.989	7.562	7.836	17.380
100	13.807	10.333	8.872	9.707	10.069
150	15.287	16.272	10.200	11.948	14.488
200	20.499	17.229	12.267	14.435	14.930
250	35.529	24.979	17.620	21.147	23.042
300	44.655	28.650	20.797	22.453	25.643
350	45.436	34.085	20.119	26.434	27.521
400	53.751	37.015	26.881	30.300	32.461
450	53.146	46.209	31.683	31.577	37.398
500	63.913	46.008	31.588	43.612	46.422

Table 8.3: Classification times in seconds for programs of increasing size when split into modules of different size for 'with cyclic mode' when testing with implicit hydrogens

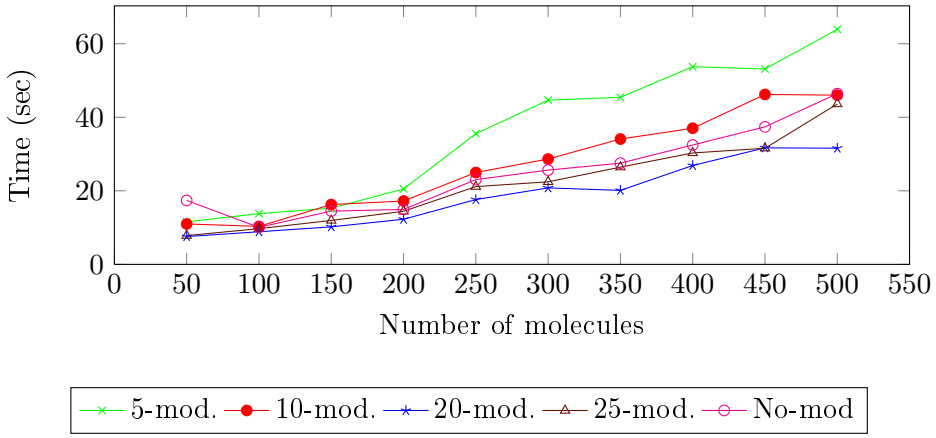


Figure 8.3: Curves of classification times for 'with cyclic' mode and testing with implicit hydrogens

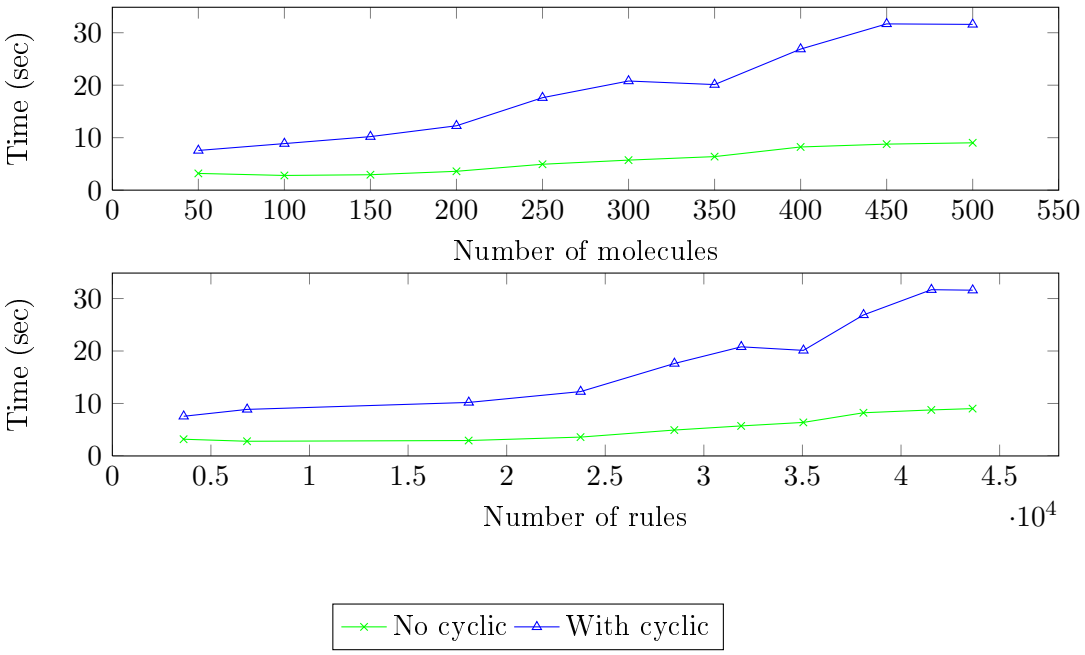


Figure 8.4: Curves of optimal classification times (module size 20)

the suite of 51 modelled chemical classes. One can observe that the rules encoding cyclicity-related classes introduce a significant overhead for the classification times. In fact, it is the class that recognises molecules with cycles of arbitrary length that incurs the greatest performance penalty. Rules r_1-r_4 in Section 7.4.4 (page 168) that encode the class of cyclic molecules need to identify patterns that are extremely frequent in molecular graphs; as a consequence, detecting ring-containing molecules is computationally expensive. However, since our class definition for cyclic molecules detects compounds with cycles of variable length, which is a significant property for the construction of chemical hierarchies, we consider this overhead acceptable.

8.3.2 Tests with Explicit Hydrogens

Similarly to before, we created rule representations of 500 molecules, with sizes ranging from 1 to 138 atoms; the higher number of atoms is due to the explicitly represented hydrogens. We also defined 30 functional group classes that are characterised by small substructures (functional groups of 2 to 8 atoms), e.g. organic hydroxy. We modelled each with two rules of the form r_{oh} and g_{oh} (page 64), using distinct pred-

icates r and g for each pair of rules; hydrogens were represented explicitly for that modelling. Moreover, we modelled the same chemical classes as in Section 8.3.1, but for some of them the definition was slightly adjusted due to the explicit statement of hydrogens. Some of the chemical classes recognition rules from Section 8.3.1 were superseded by the modelling of the functional group classes. These modifications resulted in 43 chemical classes for the 'no cyclic' mode and 46 for the 'with cyclic' mode. Similarly to the previous tests, we constructed 10 DLV programs each of which contained rule descriptions of $50 \cdot i$ different molecules, where $1 \leq i \leq 10$; in addition, we generated $50 \cdot i$ facts for the molecules (e.g. ascorbicAcid(a)) and 30 facts for the functional group classes (e.g. organicHydroxy(m)) that allowed us to compute superclasses of molecules and functional group classes, respectively. Table 8.4 provides the number of rules for each program. In contrast to the previous case, we did not execute an MSA check (as MSA is designed only for positive existential rules), but we verified manually that the assembled program was R-stratified and R-acyclic.

No of molecules	No of rules 'no cyclic'	No of rules 'with cyclic'
50	7283	7289
100	16382	16388
150	24214	24220
200	32623	32629
250	39625	39631
300	47031	47037
350	54288	54294
400	62153	62159
450	71119	71125
500	78939	78945

Table 8.4: Number of rules for programs of different size when testing with explicit hydrogens

Let $1 \leq i \leq 10$ and let P be the program containing the rules describing $50 \cdot i$ molecules, the 30 functional group classes and the 43 chemical classes ('no cyclic' mode); let also F be the set comprising the $50 \cdot i + 30$ facts of molecules and functional

group descriptions, respectively. In a first experiment and for $i = 1$, we tried to compute a stable model of $P \cup F$ using DLV, but the system failed to compute this result within a time limit of 600 seconds. In a second experiment and for i such that $1 \leq i \leq 10$, we split P into R-strata in order to consecutively compute the stable model of each R-stratum. This resulted in five R-strata of P, the first stratum P_1 of which contained the majority of the rules, while the rules of the remaining four R-strata formed a stratified program P_2^5. We thus used DLV to compute the stable model of $P_1 \cup F$, converted the result into a new set of facts S_{P}^1, and used DLV to compute the stable model of $S_{\mathsf{P}}^1 \cup P_2^5$ (we use here the notation introduced in Chapter 5 according to which S_{P}^1 is the stable model of the first R-stratum P_1 of an R-stratified program P). Detailed results for the second experiment (which was also repeated for 'with cyclic' mode and for different modularisation settings) are presented next.

We applied splitting into modules and distinction between 'no cyclic' and 'with cyclic' mode as usual; each module contained the chemical classes rules and the functional group rule pairs. Tables 8.5 and 8.6 depict the corresponding time measurements, whereas figures 8.5 and 8.6 illustrate the same results graphically. In Table 8.6, 'MEM-EX' indicates memory exhaustion, that is the program exited before completing the classification due to memory constraints.

Overall we observe that the explicit representation of hydrogens comes at the price of deteriorated performance. However, the drop in performance is counterbalanced by the additional chemical class/chemical class subsumptions that were discovered (e.g. epoxide is a carbon-containing entity). Similarly to before, computing the subsumees of the cyclicity-related classes is less tractable than the subsumees of the remaining classes; here this is further exacerbated by the presence of significantly more atoms (i.e. the hydrogens). Moreover, modularisation enabled completion of the classification process for the 'with cyclic' mode, whereas memory exhaustion was reported for the non-modularised KB. The best performances for the 'no cyclic' and 'with cyclic' mode were observed for the modularised KB with module size 25. Finally, Figure 8.7 provides the curves for the shortest classification times in the 'no cyclic' and 'with

No of molecules	5-mod.	10-mod.	20-mod.	25-mod.	No-mod.
50	3.235	2.931	3.915	3.154	2.934
100	3.926	3.380	4.391	3.27	3.092
150	5.509	4.630	7.300	4.409	4.896
200	7.323	6.204	8.839	6.084	6.036
250	8.881	7.386	11.953	6.941	7.821
300	11.282	8.870	13.606	8.32	9.568
350	12.197	10.215	17.142	9.566	12.492
400	13.876	11.645	20.588	11.567	14.724
450	15.643	13.298	20.350	12.621	15.508
500	17.543	15.140	22.585	14.497	15.644

Table 8.5: Classification times in seconds for programs of increasing size when split into modules of different size for 'no cyclic mode' and testing with explicit hydrogens

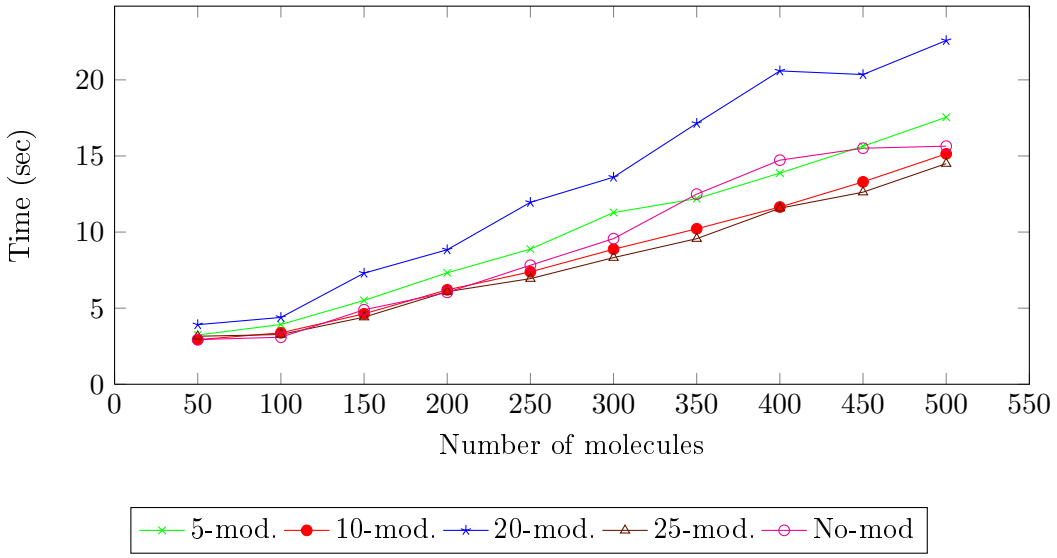


Figure 8.5: Curves of classification times for 'no cyclic' mode and testing with explicit hydrogens

No of molecules	5-mod.	10-mod.	20-mod.	25-mod.	No-mod.
50	10.031	11.394	9.229	11.427	7.191
100	19.394	18.320	16.965	16.244	15.940
150	27.879	25.711	22.801	27.447	24.807
200	42.924	32.109	33.552	33.595	35.784
250	51.758	42.352	36.921	39.880	46.966
300	61.955	47.147	50.860	46.973	55.780
350	68.026	58.610	57.867	52.234	77.321
400	74.257	65.175	66.045	56.987	84.352
450	81.307	67.974	75.047	66.554	MEM-EX
500	89.728	80.272	77.587	77.574	MEM-EX

Table 8.6: Classification times in seconds for programs of increasing size when split into modules of different size for 'with cyclic mode' and testing with explicit hydrogens

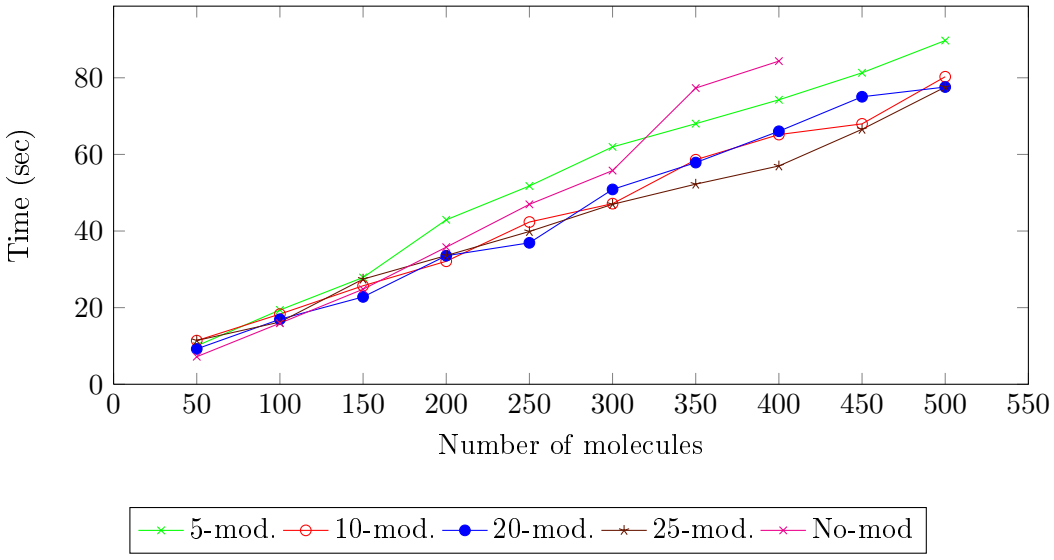


Figure 8.6: Curves of classification times for 'with cyclic' mode and testing with explicit hydrogens

cyclic' mode.

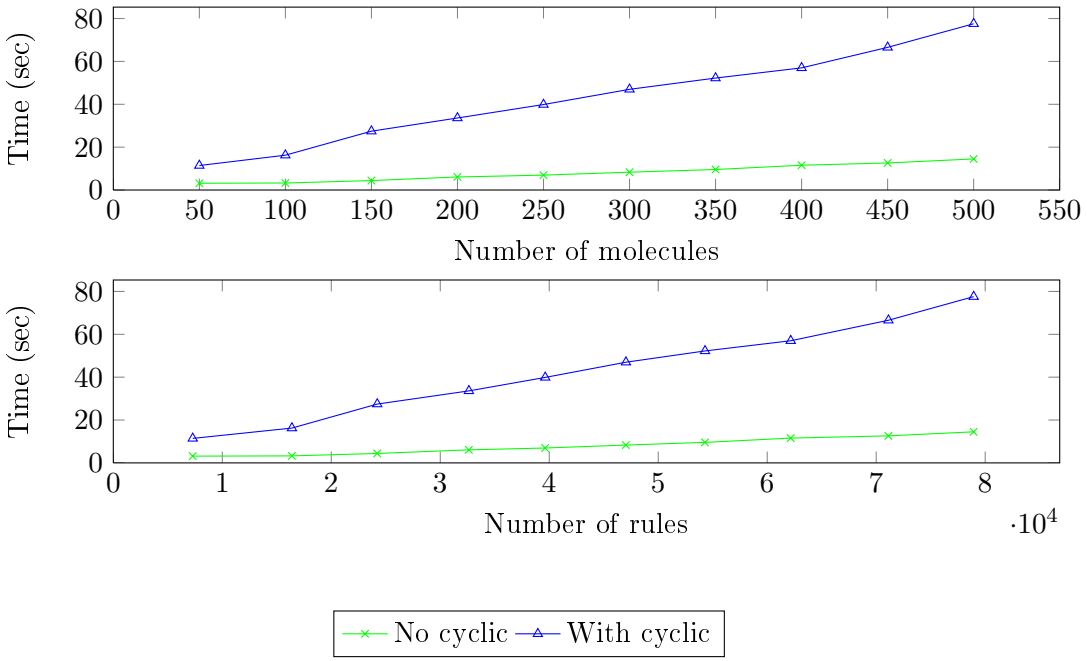


Figure 8.7: Curves of optimal classification times and testing with explicit hydrogens

8.4 Computed Subsumptions

While conducting the experiments with implicit hydrogens we computed 6,768 non-trivial subsumptions (excluding reflexive subsumptions and subsumptions with auxiliary predicates) of the form molecule/chemical class (not transitively inferred). Among them, we discovered by manual inspection a number of missing and inconsistent subclass relations from the manually curated ChEBI ontology, a few of which we present next. As one can infer from the chemical graph of ascorbic acid appearing in the top right of Figure 7.2 (page 158), ascorbic acid is a carboxylic ester as well as a polyatomic cyclic entity. In spite of the fact that these superclasses were exposed by our classification methodology, we could not identify them in the ChEBI hierarchy. Figure 8.8 shows the ancestry of ascorbic acid (ChEBI id 29073) in the OWL version of the ChEBI ontology; none of the concepts cyclic entity (ChEBI id 33595), polyatomic entity (ChEBI id 36357) or carboxylic ester (ChEBI id 33308) is among

the superclasses of ascorbic acid. Moreover, ascorbic acid is asserted as a carboxylic acid, which is not the case as can be deduced by the lack of a carboxy group in the chemical graph of ascorbic acid.

Figure 8.8: Superclasses of ascorbic acid for the ChEBI OWL ontology release 102, as illustrated by the ChEBI graph-based visualisation interface

While carrying out the experiments with explicit hydrogens we obtained 7,567 non-trivial subsumptions (excluding reflexive subsumptions and subsumptions with auxiliary predicates) of the form molecule/chemical class and chemical class/chemical class. Comparing these subclass relations with the manually created taxonomy of ChEBI also revealed omissions in ChEBI's hierarchy. For instance our prototype discovered that every organic hydroxy compound (ChEBI id 33822) is an organooxygen compound (ChEBI id 36963), which is not captured by the ancestry of organic hydroxy as specified by ChEBI (Figure 8.9). We interpret the revealing of these modelling errors as an indication of the practical relevance of our contribution.

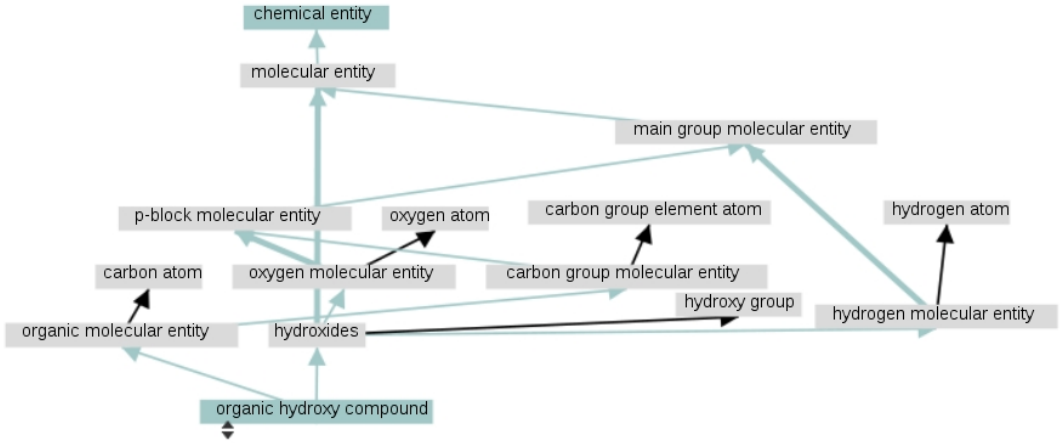


Figure 8.9: Superclasses of organic hydroxy compound for the ChEBI OWL ontology release 102, as illustrated by the ChEBI graph-based visualisation interface

8.5 Related Work and Discussion

Concerning expressive power, the current approach allows for the representation of strictly more chemical classes in comparison with other logic-based approaches to chemical classification. As we already saw in Section 8.1, Villanueva-Rosales and Dumontier [234], Konyk et al. [144] and Hastings et al. [112] proposed and tested in practice ontology-based solutions for automatic construction of chemical taxonomies. However, all these works drew upon OWL and were thus hindered by the inherent inability of OWL to represent cycles. In the current approach we are able to recognise molecules containing cycles of both arbitrary and fixed length.

Moreover, in all the previous approaches outlined above the adopted open world assumption of OWL prevented the definition of structures based on the absence of certain characteristics. In our approach we operate under the closed world assumption which permits for the definition of a broad range of chemical classes that were not expressible before, such as the class of inorganic, hydrocarbon or saturated compounds.

In terms of performance, the classification results appear more promising than previous and related work. Hastings et al. [112] reported that a total of 4 hours was required to determine the superclasses of 140 molecules, whereas LoPStER identifies

the chemical classes of 500 molecules in less than 33 seconds (three orders of magnitude improvement). Further to that, LoPStER is quicker in comparison with some preliminary experiments that we conducted [175], where the XSB logic programming engine [202] was used instead and where 450 seconds were needed to classify 70 molecules (two orders of magnitude improvement). Both cases mentioned above considered a subset of the chemical classes used here. Regarding the significant speedup in comparison with our previous work, we identify the following two factors. First, DLV is a more suitable reasoner for our setting due to its bottom-up computation strategy; on the contrary, XSB adopts a top-down evaluation strategy which required a separate test in order to discover the subsumees of each chemical class subsumer. Second, we employed a more efficient termination condition (model-summarising acyclicity [99] instead of semantic acyclicity [175]) which reduced the time needed for this part of the computation. Finally, our modularisation technique also resulted in a slight improvement of the performance times; it would be interesting in the future to investigate the behaviour of this optimisation in a more systematic way and in answer set solvers other than DLV.

Overall, we consider the following three points as the most interesting contributions of our empirical evaluation

1. We presented a prototype for logic-based classification of chemical structured entities that exhibited a significant speedup and an important expressivity advance in comparison with previous ontology-based chemical classification implementations.

2. We identified examples of missing and contradictory subsumptions from the manually curated ChEBI ontology that are present and absent, respectively, from the hierarchy computed by our prototype.

3. We demonstrated that our special condition for stratification (R-stratification) can be exploited to allow the DLV reasoner to scale to the large number of rules of our experiments and thus dramatically increase the performance of existing answer set programming engines.

For the future it would be interesting to further apply our framework towards supporting classification of other complex structures. For instance, one can exploit the expressive power of rules to represent biochemical processes and infer useful relations about them. Figure 8.10 depicts a labelled graph abstraction of a chemical reaction example discussed by Bölling et al. [29]. The process consists of parts that are arbitrarily interconnected and can thus be naturally modelled using our formalism. In the same vein, our methodology could provide rigorous definitions for the representation of lipid molecules, which are entities that can be systematically classified on the basis of their constituent functional groups. Low et al. [162] introduced the OWL DL Lipid Ontology which contains semantically explicit lipid descriptions. One could achieve more accurate modelling by casting lipids in nonmonotonic existential rules that are able to capture frequent cyclic patterns in a concise way; for example, Figure 8.11 illustrates a description graph for jasmonic acid, which is one of the lipids encountered in the abovementioned OWL ontology.

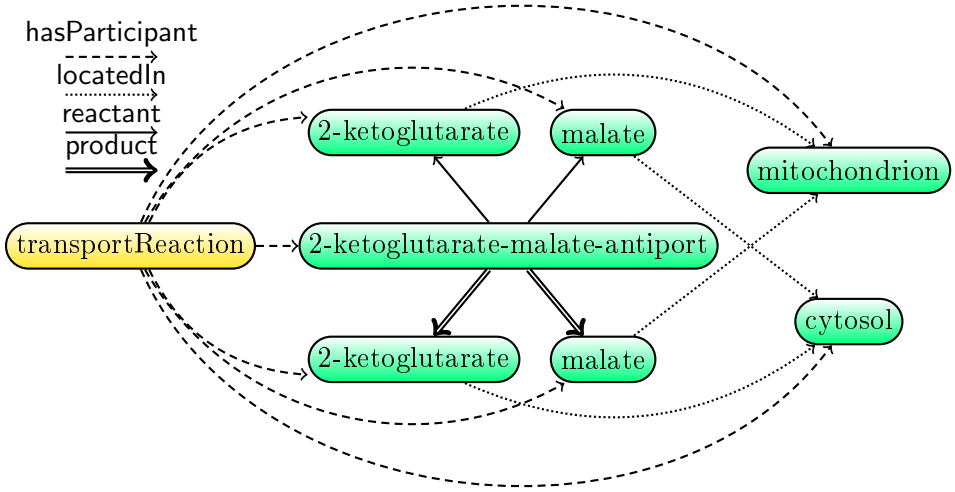


Figure 8.10: Labelled graph describing a transport reaction.

Another potential application is the use of nonmonotonic existential rules as an

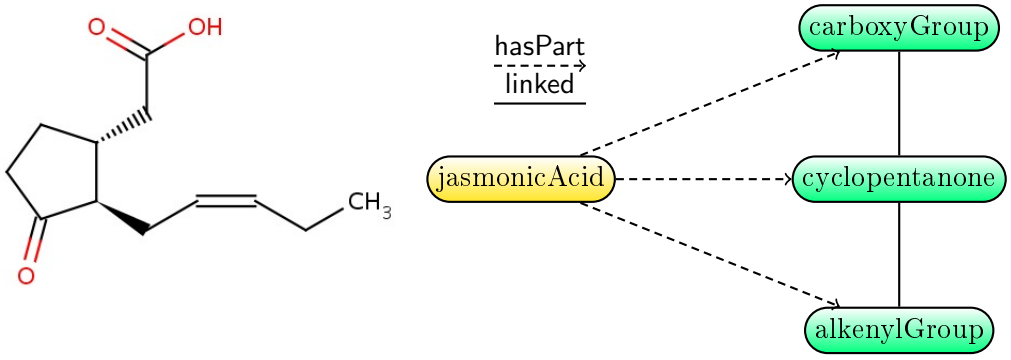


Figure 8.11: Chemical graph of jasmonic acid (left) and labelled graph abstracting jasmonic acid based on the functional groups partonomy (right).

ontology language within the 'Digital Aristotle' project,[4] a joint academic and industrial effort that aims to develop a conceptual model for chemistry, biology and physics via formalisation of university textbook knowledge [106]. Our formalism could aid in the modelling of highly interconnected objects found in that context such as the biological concept of mitosis or the eukaryotic cell structure [54].

[4] http://www.projecthalo.com/

Chapter 9

Outlook

In this final chapter, we review the scope of the thesis (Section 9.1), we sum up the major results and we discuss their significance (Section 9.2). We conclude by outlining a number of suggestions for future research (Section 9.3).

9.1 Thesis Overview

The objective of this thesis was to explore ontology languages suitable for modelling structured domains and to demonstrate that nonmonotonic existential rules can serve as a suitable basis for such languages.

To this end, we provided a theoretical and practical framework for the representation of structured entities that draws upon nonmonotonic existential rules and exhibits a favourable balance between expressive power and computational as well as empirical tractability. In order to ensure decidability of reasoning, we introduced a number of acyclicity criteria that strictly generalise many of the existing ones. We also presented a novel stratification condition that properly extends 'classical' stratification and allows for capturing both definitional and conditional aspects of complex structures. The applicability of our formalism was supported by a prototypical implementation, which was tested over a realistic knowledge base and which outperformed by up to three orders of magnitude previous evaluation efforts; additionally, our experiments identified missing and incorrect subsumptions of the manually curated

ChEBI classification.

In this work, we only considered logics for constructing knowledge bases with models of finite size, as opposed to other languages that also allow infinitely large but finitely representable models. The reason is that representing objects that are cyclic and at the same time of arbitrarily large size typically leads to undecidability for the main reasoning tasks of the underlying formalisms. Since our initial motivation was to perform classification for cyclic objects of bounded size, our investigations were focused on languages that guarantee finiteness of the stable models. Since our target domain mostly consisted of objects with a uniquely defined structure, we explored techniques for ensuring uniqueness of the stable model as well.

9.2 Summary and Impact of Results

The major results of this thesis are presented and their significance is discussed next.

9.2.1 Nonmonotonic Existential Rules as a Description Language

After surveying existing ontological formalisms, and concluding that none of them met the modelling needs of our motivating scenarios, we proposed a new framework based on nonmonotonic existential rules and showed how this framework enables the definition and recognition of concepts based on a wide range of structural features.

9.2.2 Acyclicity Conditions

Various acyclicity conditions were identified to ensure decidability of reasoning for nonmonotonic existential rules. Initially, MFA was formulated as an acyclicity condition that guarantees model finiteness for positive existential rules; it was proved that MFA is a very general condition that strictly subsumes most of the existing acyclicity criteria. However, as MFA is computationally expensive to check, we additionally formulated MSA, a slightly weaker condition that is easy to check. It was shown that MSA is still more general than many of the known acyclicity conditions. In order to accommodate rules with nonmonotonic negation in the body, R-acyclicity was in-

troduced. R-acyclicity was further extended by exploiting constraints that take into account the shape of the input facts. Tight complexity results were established for checking all of the acyclicity conditions described above as well as for reasoning over sets of rules satisfying the previously mentioned conditions.

MSA and MFA have the distinguishing advantage of being formulated as semantic conditions, which implies that they can be checked using any sound and complete logic programming engine. Further to that, MFA has the additional benefit of allowing to merge the acyclicity check with the computation of the model: if the set of rules is MFA, then the model can be extracted from the set of facts that was derived to perform the check. In addition to supporting sound and complete reasoning procedures for nonmonotonic existential rules, acyclicity conditions are also of interest to a broad variety of fields including:

- Verification of knowledge action bases. Hariri et al. employ acyclicity conditions to enable verification of temporal properties over DL knowledge bases enriched with information about actions [108].

- Implementation of logic programming engines. Answer set solvers often perform an acyclicity check on the input logic programs to ensure that inputs which give rise to stable models of infinite size are ruled out. Examples include ω-acyclicity, λ-acyclicity and argument-restrictedness that have been incorporated in the SMODELS [227], GRINGO [89] and DLV [153] logic programming systems, respectively.

- Finiteness of target schema instances in data exchange. Positive existential rules can be used to capture tuple-generating dependencies, which are rules used in data exchange specifying how to translate data structured under a source schema into data conforming to a target schema. To this end, acyclicity conditions can be used to guarantee finiteness of the target instance and thus feasibility of the transformation [80].

- Query answering over DL knowledge bases. Conjunctive query answering over DL ontologies can be of high computational complexity, such as query answer-

ing over Horn-$\mathcal{SHOIQ}$ ontologies which is 2ExpTime-complete [191]. One approach to improve the efficiency of query execution is to adopt the so-called combined approach, that is to rewrite both the query and the knowledge base [143]. Other practical applications solve this problem using materialisation, where ontology consequences are precomputed using chase and stored in a semantic data store for subsequent query evaluation; examples of such systems include Oracle's Semantic Data Store [249], Sesame [37], Jena [49], OWLim [139] and DLE-Jena [179]. Since this approach requires termination of the chase, materialisation is usually restricted to OWL 2 RL axioms (i.e. datalog rules), which can lead to missing answers. In spite of the fact that there are systems, such as OWLim [27] and Jena, that partially support existential rules, such support typically fails to provide any completeness or termination guarantees. Acyclicity conditions can be exploited to address these issues, since a complete materialisation of an acyclic ontology can be computed without the risk of non-termination. In a recent evaluation over a corpus of 336 DL ontologies, 63.8% of them were found to be MSA [100] which suggests that acyclicity conditions can be used to enable materialisation-based query answering beyond the OWL 2 RL profile.

In addition to designing new acyclicity conditions, we revisited many of the existing ones and compared them both with each other and with our own conditions. We thus brought together and clarified the relationship between several acyclicity criteria originating from a variety of backgrounds such as databases, logic programming and rule-based knowledge representation. The map that was charted associating acyclicity conditions w.r.t. complexity of checking and expressive power can act as a useful guide to researchers across all the fields outlined above.

9.2.3 Stratification Conditions

A novel stratification condition, R-stratification, that ensures uniqueness of the stable model for nonmonotonic existential rules was presented and it was shown that R-stratification properly extends classical stratification. A strengthened version of R-

stratification was further defined by taking into account a set of constraints that reflects restrictions on the input facts. Upper and lower complexity bounds were proved for testing the two stratification conditions as well as for reasoning over R-stratified and R-acyclic sets of rules, both with and without constraints. The benefits arising from introducing R-stratification are manifold.

- First, as we explained in Section 5.1, R-stratification permits one to encode and reason about both definitional and conditional aspects of classes depending on structural features. It thus yields a powerful ontology language for modelling complex structures.

- Second, R-stratification specifies a way to augment existential rules with non-monotonic negation under stable model semantics while retaining the same complexity for the basic reasoning tasks, i.e. 2ExpTime-completeness for combined complexity and P-completeness for data complexity. This is a result of general interest, since extending existential rules with negation is an active research topic due to the applicability of such rules to tasks such as query answering of ontological knowledge bases [96] and of databases [116].

- Third, our stratification notion could be exploited to optimise the performance of answer set programming engines as it could serve as a tool to guide the evaluation order among the rules of the program. This is strongly evidenced by our experiences with the DLV reasoner, where we failed to compute the stable model of an R-stratified program; however, when we partitioned the same program into R-strata and calculated the stable model of each R-stratum separately, the computation was completed in a few seconds. Further to that, since deciding R-stratification was proved to be coNP-complete w.r.t. the size of each rule separately, it is expected that checking R-stratification in practice should not be prohibitely expensive.

9.2.4 Practical Considerations

In order to facilitate the use of our ontology language by application experts the DGLP syntax was designed, which is an intermediate representation language for structured entities closer to natural language than to first-order logic notation. To this end, the shapes of the most frequently used rules were identified, a BNF grammar was presented for them and mappings of DGLP expressions to nonmonotonic existential rules were defined. The objective of that effort was to equip ontology developers with a 'less logician-like' syntax in the same path as the OWL Manchester syntax—a language that has successfully aided non-logicians to write OWL ontologies in an intuitive way.

The DGLP syntax can be of great benefit to domain specialists, as it is designed to help them read and write nonmonotonic existential rules in a quick and easy way. In fact, discussions with EBI researchers revealed that such a syntax should be an essential component of any ontology editor. Moreover, DGLP syntax intentionally follows as faithfully as possible the OWL Manchester syntax in order to exploit familiarity that ontology developers might already have with it.

In order to assess our framework in practice, an empirical evaluation over a biochemical knowledge base was carried out. Due to lack of knowledge bases consisting of rules, the ChEBI database and ontology was selected as a starting point to create a pragmatic knowledge base: molecular graph files retrieved from the ChEBI database were programmatically converted into rules and ChEBI chemical classes with human-readable but not machine-processable descriptions were manually formalised as rules. LoPStER , a prototype that performs logic-based computation of subsumptions over the constructed knowledge base, was developed. LoPStER implements the reasoning procedure for rules that was presented earlier by relying on the DLV logic programming engine for stable model computation.

Two series of experiments were conducted, in the first of which subsumptions of the form molecule/chemical class were identified and in the second of which subclass relations of the form molecule/chemical class and chemical class/chemical class were computed. In the former setting, 500 molecules were classified under 51 chemical

classes in less than 32 seconds, whereas in the latter 500 molecules and 30 chemical classes were classified under 76 chemical classes in 78 seconds. A significant speedup was exhibited in comparison with previous ontology-based chemical classification implementations, such as the one that was based on OWL and Description Graphs [112]. Moreover, in our case a much broader—in terms of expressive power—range of chemical classes was axiomatised. Further to that, numerous missing and contradictory subsumptions from the manually curated ChEBI ontology were identified that were present and absent, respectively, from the hierarchy computed by our prototype. The experimental results described above are interesting for a number of reasons.

- First, it was demonstrated that reasoning about non-tree-shaped entities using nonmonotonic existential rules is practically feasible, especially given the availability of numerous highly optimised logic programming reasoners.

- Second, our prototype could form the basis of a logic-based application to assist biocurators of the ChEBI ontology towards the sanitisation and the enrichment of the existing chemical taxonomy. Similarly, such a tool could contribute to a more rapid development of the ChEBI ontology and to the efforts of the ChEBI team to make annotated chemical datasets available to the public. We are currently in communication with ChEBI developers for the design and implementation of a software framework that would serve that purpose.

- Third, our experiments revealed a novel application of logic programming engines that have typically been oriented towards solving intractable search problems. In fact, our chemical classification problem was selected to form part of the problem suite for the fourth answer set programming competition,[1] which is a venue for comparing the performance of answer set solvers by testing them over a range of diverse problems with varying degrees of difficulty.

[1] `https://www.mat.unical.it/aspcomp2013/ChemicalClassification`

9.3 Future Directions

The work presented in this thesis settles only some of the issues related to representation of structures; it could thus be extended in several ways, both with a theoretical and a practical focus. Some suggestions related to implementation are provided next.

- The most pressing need is to provide better tool support to ontology developers for modelling structured domains with nonmonotonic existential rules. That would imply the development of an ontology editor with a surface syntax parser that would allow for the creation of DGLP expressions and their automatic conversion into nonmonotonic existential rules. Currently LoPStER is only designed to handle rules in the DLV syntax and chemical table files stored as input files; a future system should be able to accept input axioms provided via a user interface. Moreover, LoPStER only performs an MSA check over the provided set of rules; it would be useful to also implement and incorporate in the system other checks introduced in this thesis, such as MFA, R-acyclicity and R-stratification; in particular for the last two conditions, an efficient algorithm for checking positive and negative reliance between rules needs to be implemented first. Given that the sets of rules provided by the user might be found to be non-acyclic or non-stratified, it would be interesting to explore justification services that would guide ontology engineers through changing their knowledge base into acyclic or stratified ones; such services have a long tradition in ontology environments and have been proven to be useful in practice [122]. Moreover, it would be practically relevant to turn such a tool into a plug-in for Protégé [142] in order to maximise the outreach of our framework to the ontology engineering community; it could also be of interest to explore integration with life sciences platforms [222] or chemical structure visualisation tools [1, 145].

- It would be useful to define a mapping of the introduced formalism to RDF [141] in order to formalise the exchange of rule-based ontologies; the RDF Concrete Syntax of SWRL rules could provide a basis for such an effort [129].

- More extensive testing of LoPStER over other rule-based ontologies is needed.

Due to lack of test data, our experiments were centered around the ChEBI ontology, but it would be equally interesting to cast OWL ontologies describing complex structures (such as GALEN [204] and FMA [212] but also see Section 8.5) into our framework and carry out classification experiments. An interesting technique about how this could be done is provided by Motik et al. [184].

- It would be promising to integrate our reliance approach into existing rule engines and investigate the impact that this has on their performance.

A number of suggestions to address open issues arising from the theoretical contributions presented in this thesis follow.

- Devise algorithms that given a non-R-stratified program identify a suitable set of constraints such that the program becomes R-stratified under them.
- Clarify the relationship between R-stratification and weak stratification.
- Extend MSA and MFA so that they become applicable to rules with nonmonotonic negation in the body.
- Appropriately modify MFA so that it marks as cyclic a program whose chase contains a term with a function symbol nested more than once as opposed to a term with a function symbol nested once which is the case for the current definition.
- Compare FD, AR, Γ-acyclicity and R-acyclicity for rules that contain negative bodies.
- Identify complexity bounds for reasoning over R-acyclic and/or R-stratified programs that are ∃-1 or have bounded predicate arity.

A number of more foundational problems could be explored to obtain an even more expressive formalism for structured domains.

- One could extend acyclicity and stratification conditions to take into account rules that contain equality in the head; this could be a technically involved problem as such rules usually lead to numerous additional reliances and more sophisticated mechanisms need to be devised in order to treat such effects; a first step towards analysing the effects of equality when combined with nonmonotonic negation appears in Krötzsch et al. [148]. A use case where rules with equality in the head can be useful is discussed at the end of Section 7.4.1.

- Another natural direction would be to extend nonmonotonic existential rules with disjunction in the head so as to enable the encoding of disjunctive information; there have already been works investigating the impact of disjunction on existential rules [30, 7]—it would be interesting to study what happens when this is combined with nonmonotonic negation.

- A further promising extension would be to investigate how nonmonotonic existential rules could be extended with concrete domains in order to capture definitions depending on numerical values [166, 167, 168]; that could allow for representation of descriptors that are much needed in life sciences domains, examples of which include molecular weights or angles of chemical bonds.

- Our formalism is well-suited for the specification of structures that are of fixed size; an important challenge is to extend it so that it can represent structures that are arbitrarily large but of bounded size such as molecules consisting of an unspecified number of fused rings.

- Finally, it remains open how one could use a logic-based formalism to capture classes based on advanced structural features, such as compounds consisting of exactly n rings of arbitrary size (where $n \geq 1$) or cyclic compounds with ring atoms belonging to a particular element.

From the discussion above, it follows that our work has opened up a whole range of problems in the intersection of logic and knowledge representation that are worth exploring. Overall, in the course of this thesis we laid the theoretical foundation of

a sound and complete as well as expressive ontology language for the representation of structures that at the same time proved promising in practice. We thus made a step forward in advancing the state of the art of knowledge representation technologies. From a modelling point of view, our approach could stimulate the adoption of a different and expressive reasoning paradigm for which highly optimised mature reasoners are available; it could thus pave the way for the representation of a broader spectrum of knowledge about the world.

Appendix A

Definition of DGLP Syntax

A.1 BNF Rules for DGLP Syntax

For the following BNF derivation rules, terminal symbols appear surrounded by single quotes and non-terminal symbols are shown in bold face; symbols that may appear zero or one time (optional) are enclosed in square brackets ([...]) and symbols that may appear zero or more times are enclosed in curly brackets ({...}); the vertical bar (|) indicates an alternative choice and an expression of the form $\mathsf{s}_1..\mathsf{s}_n$ denotes choice between the elements of the interval from s_1 to s_n inclusive. For Table A.1 we require that for each **DescriptionGraph** the nodes are provided sequentially w.r.t. the node identifier and that the node identifiers **NI** used in the definition of each **Edge** should occur in the definition of a **Node** of the same **DescriptionGraph** ; similarly, for each **DescriptionGraphNegLabels** expression. Finally, for Table A.1 we require that $B \rightarrow H$ is an expression of the form (2.1).

UppercaseLetter	::=	'A' .. 'Z'
LowercaseLetter	::=	'a' .. 'z'
Letter	::=	**UppercaseLetter** | **LowercaseLetter**
Digit	::=	'0' .. '9'
Alphanumeric	::=	**Letter** | **Digit**
LowercaseString	::=	**LowercaseLetter**{**Alphanumeric**}
NaturalNumber	::=	**Digit**{**Digit**}
ClassName	::=	**LowercaseString**
PropertyName	::=	**LowercaseString**
NI	::=	**NaturalNumber**
Literal	::=	**ClassName** | 'NOT' **ClassName**
Ontology	::=	**DGLP** {, **DGLP** }
DGLP	::=	**Rule** | **Implies** | **ImpliedBy** | **ImpliesAndImpliedBy**
Rule	::=	'NERule' '(' $B \rightarrow H$ ')'
Implies	::=	**ClassName** 'SubClassOf'
		{ **ClassName** 'AND' } **Consequent**
Consequent	::=	**Property** 'SOME' **DescriptionGraph**
Property	::=	**PropertyName** | 'Inverse' '(' **PropertyName** ')'
DescriptionGraph	::=	'Graph' '(' **NodeSet** **EdgeSet** ')'
NodeSet	::=	'Nodes' '(' **Node** {, **Node** } ')'
Node	::=	**NI** { **ClassName** }
EdgeSet	::=	'Edges' '(' [**Edge** {, **Edge** }] ')'
Edge	::=	**NI** **NI** **PropertyName** { **PropertyName** }

Table A.1: BNF rules for the definition of structured objects

ImpliedBy	::=	**ClassImpliedBy** | **PropertyImpliedBy**
ClassImpliedBy	::=	**ClassName** 'SuperClassOf'
		Antecedent { 'OR' **Antecedent** }
Antecedent	::=	**ClassExpression**
		| [**ClassExpression** 'AND'] **Predicate**
ClassExpression	::=	**ClassName** { 'AND' **Literal** }
Predicate	::=	**Existential** | **CardinalityRestriction** | **Universal**
Existential	::=	**Property** 'SOME' [**Predicative**]
Predicative	::=	**ClassExpression** | **DescriptionGraphNegLabels**
DescriptionGraphNegLabels	::=	'GraphNL' '(' **NodeSetNL** **EdgeSetNL** ')'
NodeSetNL	::=	**NodesMode** '(' **NodeNL** {, **NodeNL** } ')'
NodesMode	::=	'Nodes' | 'DisjointNodes'
NodeNL	::=	**NI** { **Literal** }
EdgeSetNL	::=	'Edges' '(' [**EdgeNL** {, **EdgeNL** }] ')'
EdgeNL	::=	**NI** **NI** **PropertyName**
		{ **PropertyName** } { 'NOT' **PropertyName** }
PropertyImpliedBy	::=	**PropertyName** 'SuperPropertyOf'
		PropertyExpression { 'OR' **PropertyExpression** }
PropertyExpression	::=	**PropertyName** { 'AND' **PropertyName** }
		{ 'AND' 'NOT' **PropertyName** }
		| **PropertyName** 'o' **PropertyName**
		{ 'o' **PropertyName** }
		| 'Inverse' '(' **PropertyName** ')'

Table A.2: BNF rules for recognition of classes and properties

NonZeroDigit	::=	'1' .. '9'
NonZeroNaturalNumber	::=	**NonZeroDigit** {Digit}
N	::=	**NonZeroNaturalNumber**
ZN	::=	**NaturalNumber**
CardinalityRestriction	::=	**Property** 'AT LEAST' **N** [**Predicative**]
		| **ClassName** 'AND' **Property** 'AT MOST'
		ZN [**Predicative**]
		| **Property** 'EXACTLY' **N** [**Predicative**]
Universal	::=	**ClassExpression** 'AND' **Property** 'ONLY' **Disjunction**
Disjunction	::=	'(' **Literal** { 'OR' **Literal** } ')'
ImpliesAndImpliedBy	::=	**ClassName** 'EquivalentTo'
		Consequent { 'AND' **ClassName** }

Table A.3: BNF rules for cardinality constraints and equivalence axioms

A.2 Mappings of DGLP Axioms to Nonmonotonic Existential Rules

The mapping of the following translation rules is defined recursively, that is in certain cases the mapping of a construct relies on the mapping of its subconstruct(s), where the logical expression generated by the recursive invocation of the mapping is placed in the position of the recursive invocation. Additionally, the auxiliary operator HA is used to specify that the elements upon which HA operates shall be translated into head atoms; the operator BA is analogous for body atoms.

Element E of the BNF grammar	Logical expression NER(E)
DGLP$_1$, . . . , **DGLP**$_n$	$\{\text{NER}(\textbf{DGLP}_i)\}_{i=1}^{n}$
ClassName SubClassOf **ClassName**$_1$ AND . . . AND **ClassName**$_n$ AND **Consequent**	**ClassName** (x) $\rightarrow \bigwedge_{i=1}^{n}$ **ClassName**$_i$ (x) $\wedge$ HA(**Consequent**)

Element E of the BNF grammar	Logical expression HA(E)
PropertyName SOME Graph (Nodes (**Node**$_1$, . . . , **Node**$_n$) **EdgeSet**)	$\exists_{i=1}^{n} y_i . \bigwedge_{i=1}^{n}$ **PropertyName** (x, y_i) $\wedge \bigwedge_{i=1}^{n}$HA(**Node**$_i$) $\wedge$ HA(**EdgeSet**)
Inverse(**PropertyName**) SOME Graph (Nodes (**Node**$_1$, . . . , **Node**$_n$) **EdgeSet**)	$\exists_{i=1}^{n} y_i . \bigwedge_{i=1}^{n}$ **PropertyName** (y_i, x) $\wedge \bigwedge_{i=1}^{n}$HA(**Node**$_i$) $\wedge$ HA(**EdgeSet**)
(**NI** **ClassName**$_1$ . . . **ClassName**$_n$)	$\bigwedge_{i=1}^{n}$ **ClassName**$_i$ (y_{NI})
Edges (**Edge**$_1$, . . . , **Edge**$_n$)	$\bigwedge_{i=1}^{n}$HA(**Edge**$_i$)
(**NI**1 **NI**2 **PropertyName**$_1$ . . . **PropertyName**$_n$)	$\bigwedge_{i=1}^{n}$ **PropertyName**$_i$ (y_{NI1}, y_{NI2})

Table A.4: Mappings to NER for **Implies** expressions; we have $n \geq 1$ where applicable

Element E of the BNF grammar	Logical expression NER(E)
ClassName SuperClassOf **Antecedent**$_1$ OR ... OR **Antecedent**$_n$	BA(**Antecedent**$_i$) $\rightarrow$ **ClassName** (x) $i = 1, \ldots, n$
NERule ($B \rightarrow H$)	$B \rightarrow H$
Element E of the BNF grammar	Logical expression BA(E)
ClassName$_1$ AND ... AND **ClassName**$_m$ AND NOT **ClassName**$_{m+1}$ ... AND NOT **ClassName**$_n$	$\bigwedge_{i=1}^{m}$ **ClassName**$_i$ (x) $\wedge$ $\bigwedge_{i=m+1}^{n}$ **not ClassName**$_i$ (x)
ClassExpression AND **Predicate**	BA(**ClassExpression**) $\wedge$ BA(**Predicate**)
PropertyName SOME **ClassName**$_1$ AND ... AND **ClassName**$_k$ AND NOT **ClassName**$_{k+1}$ ... AND NOT **ClassName**$_\ell$	**PropertyName** (x, z) $\wedge$ $\bigwedge_{i=1}^{k}$ **ClassName**$_i$ (z) $\wedge$ $\bigwedge_{i=k+1}^{\ell}$ **not ClassName**$_i$ (z)
Inverse(**PropertyName**) SOME **ClassName**$_1$ AND ... AND **ClassName**$_k$ AND NOT **ClassName**$_{k+1}$ ... AND NOT **ClassName**$_\ell$	**PropertyName** (z, x) $\wedge$ $\bigwedge_{i=1}^{k}$ **ClassName**$_i$ (z) $\wedge$ $\bigwedge_{i=k+1}^{\ell}$ **not ClassName**$_i$ (z)

Table A.5: Mappings to NER for **ImpliedBy** expressions; we have $n \geq 1$, $n \geq m \geq 1$ and $\ell \geq k \geq 0$ where applicable

Element E of the BNF grammar	Logical expression BA(E)
PropertyName SOME GraphNL (Nodes (**NodeNL**$_1$, . . . , **NodeNL**$_\mathbf{n}$) **EdgeSetNL**)	$\bigwedge_{i=1}^{n}$ **PropertyName** $(x, z_i) \wedge$ $\bigwedge_{i=1}^{n}$BA(**NodeNL**$_\mathbf{i}$) $\wedge$ BA(**EdgeSet**)
Inverse(**PropertyName**) SOME GraphNL (Nodes (**NodeNL**$_1$, . . . , **NodeNL**$_\mathbf{n}$) **EdgeSetNL**)	$\bigwedge_{i=1}^{n}$ **PropertyName** $(z_i, x) \wedge$ $\bigwedge_{i=1}^{n}$BA(**NodeNL**$_\mathbf{i}$) $\wedge$ BA(**EdgeSetNL**)
PropertyName SOME GraphNL (DisjointNodes (**NodeNL**$_1$, . . . , **NodeNL**$_\mathbf{n}$) **EdgeSetNL**)	$\bigwedge_{1 \le i < j \le n} z_i \neq z_j \wedge$ $\bigwedge_{i=1}^{n}$ **PropertyName** $(x, z_i) \wedge$ $\bigwedge_{i=1}^{n}$BA(**NodeNL**$_\mathbf{i}$) $\wedge$ BA(**EdgeSetNL**)
Inverse(**PropertyName**) SOME GraphNL (DisjointNodes (**NodeNL**$_1$, . . . , **NodeNL**$_\mathbf{n}$) **EdgeSetNL**)	$\bigwedge_{1 \le i < j \le n} z_i \neq z_j \wedge$ $\bigwedge_{i=1}^{n}$ **PropertyName** $(z_i, x) \wedge$ $\bigwedge_{i=1}^{n}$BA(**NodeNL**$_\mathbf{i}$) $\wedge$ BA(**EdgeSetNL**)
(**NI** **ClassName**$_1$ . . . **ClassName**$_\mathbf{m}$ NOT **ClassName**$_{\mathbf{m}+1}$ . . . NOT **ClassName**$_\mathbf{n}$)	$\bigwedge_{i=1}^{m}$ **ClassName**$_\mathbf{i}$ $(z_{NI}) \wedge$ $\bigwedge_{i=m+1}^{n}$ **not** **ClassName**$_\mathbf{i}$ (z_{NI})
Edges (**EdgeNL**$_1$, . . . , **EdgeNL**$_\mathbf{n}$)	$\bigwedge_{i=1}^{n}$BA(**EdgeNL**$_\mathbf{i}$)
(**NI**1 **NI**2 **PropertyName**$_1$ . . . **PropertyName**$_\mathbf{m}$ NOT **PropertyName**$_{\mathbf{m}+1}$ . . . NOT **PropertyName**$_\mathbf{n}$)	$\bigwedge_{i=1}^{m}$ **PropertyName**$_\mathbf{i}$ $(z_{NI1}, z_{NI2}) \wedge$ $\bigwedge_{i=m+1}^{n}$ **not** **PropertyName**$_\mathbf{i}$ (z_{NI1}, z_{NI2})

Table A.6: Mappings to NER for **DescriptionGraphNegLabels** expressions; we have $n \geq 1$ and $n \geq m \geq 1$ where applicable

Element E of the BNF grammar	Logical expression NER(E)
PropertyName SuperPropertyOf **PropertyExpression**$_1$ OR ... OR **PropertyExpression**$_n$	BA(**PropertyExpression**$_i$) $\rightarrow$ **PropertyName** (x_0, x_n) $i = 1, \ldots, n$
Element E of the BNF grammar	Logical expression BA(E)
PropertyName$_1$ AND ... AND **PropertyName**$_m$ AND NOT **PropertyName**$_{m+1}$ ... AND NOT **PropertyName**$_n$	$\bigwedge_{i=1}^{m}$ **PropertyName**$_i$ $(x_0, x_n) \wedge$ $\bigwedge_{i=m+1}^{n}$ **not PropertyName**$_i$ (x_0, x_n)
PropertyName$_1$ o ... o **PropertyName**$_n$	$\bigwedge_{i=1}^{n}$ **PropertyName**$_i$ (x_{i-1}, x_i)
'Inverse' '(' **PropertyName** ')'	**PropertyName** (x_n, x_0)

Table A.7: Mappings to NER for **PropertyImpliedBy** expressions; we have $n \geq 1$ and $n \geq m \geq 1$ where applicable

Element E of the BNF grammar	Logical expression NER(E)
ClassExpression AND **PropertyName** ONLY (**ClassName**$_1$ OR ... **ClassName**$_m$ OR NOT **ClassName**$_{m+1}$ OR ... NOT **ClassName**$_n$)	BA (**ClassExpression**) $\wedge$ **not AuxClassName** (x) and also add the following rule **PropertyName** $(x, z) \wedge$ **not ClassName**$_1$ $(z) \wedge \ldots \wedge$ **not ClassName**$_m$ $(z) \wedge$ **ClassName**$_{m+1}$ $(z) \wedge \ldots \wedge$ **ClassName**$_n$ (z) $\rightarrow$ **AuxClassName** (x)

Table A.8: Mappings to NER for **Universal** expressions; we have $m \geq 0$ and $n \geq 0$ where applicable

Element E of the BNF grammar	Logical expression BA(E)
Property AT LEAST **N** **CN**$_1$ AND ... **CN**$_\mathsf{k}$ AND NOT **CN**$_{\mathsf{k}+1}$ AND ... NOT **CN**$_\ell$	BA (**Property** SOME GraphNL (DisjointNodes (1 **CN**$_1$ ... **CN**$_\mathsf{k}$ NOT **CN**$_{\mathsf{k}+1}$ ... NOT **CN**$_\ell$, ..., **N** **CN**$_1$ ... **CN**$_\mathsf{k}$ NOT **CN**$_{\mathsf{k}+1}$ ... NOT **CN**$_\ell$) Edges ( )))
Property AT LEAST **N** GraphNL (**NodesMode** (NI$_1$ **NodeLabel**$_1$, ..., NI$_\mathsf{m}$ **NodeLabel**$_\mathsf{m}$) Edges (S$_1$ D$_1$ **EdgeLabel**$_1$, ..., S$_\ell$ D$_\ell$ **EdgeLabel**$_\ell$))	BA (**Property** SOME GraphNL (DisjointNodes (**NI**$^1_1$ **NodeLabel**$_1$,..., **NI**$^1_\mathsf{m}$ **NodeLabel**$_\mathsf{m}$,..., **NI**$^\mathsf{N}_\mathsf{m}$ **NodeLabel**$_\mathsf{m}$) Edges (S$^1_1$ D$^1_1$ **EdgeLabel**$_1$,..., S$^1_\ell$ D$^1_\ell$ **EdgeLabel**$_\ell$,..., S$^\mathsf{N}_\ell$ D$^\mathsf{N}_\ell$ **EdgeLabel**$_\ell$)))
ClassName AND **Property** AT MOST **ZN** **Predicative**	BA (**ClassName** AND NOT **AuxClassName**) and also add the following rule NER (begin **AuxClassName** SuperClassOf **Property** AT LEAST **ZN** + 1 **Predicative** end)
Property EXACTLY N **Predicative**	BA (**Property** AT LEAST N **Predicative** AND NOT **AuxClassName**) and also add the following rule NER (begin **AuxClassName** SuperClassOf **Property** AT LEAST **ZN** + 1 **Predicative** end)

Table A.9: Mappings to NER for **CardinalityRestriction** expressions; we have $\mathsf{k} \geq 0$, $\mathsf{m} \geq 1$ and $\ell \geq 0$ where applicable, **AuxClassName** is a fresh **ClassName** and **ZN** + 1 is **ZN** increased by 1

Element E_{def} of the BNF grammar
ClassName EquivalentTo **PropertyName** SOME Graph (Nodes (NI_1 $\mathbf{NodeLabel}^1_1$ ... $\mathbf{NodeLabel}^1_{\mathsf{k}_1}$, ..., NI_ℓ $\mathbf{NodeLabel}^\ell_1$ ... $\mathbf{NodeLabel}^\ell_{\mathsf{k}_\ell}$) Edges (S_1 D_1 $\mathbf{EdgeLabel}^1_1$ ... $\mathbf{EdgeLabel}^1_{\mathsf{m}_1}$, ..., S_n D_n $\mathbf{EdgeLabel}^\mathsf{n}_1$ ... $\mathbf{EdgeLabel}^\mathsf{n}_{\mathsf{m}_\mathsf{n}}$)) AND $\mathbf{ClassName}_1$ AND ... AND $\mathbf{ClassName}_\mathsf{h}$

Table A.10: **ImpliesAndImpliedBy** expression; we have $\ell \geq 1$, $\mathsf{n} \geq 0$, $\mathsf{k_i} \geq 0$ for $1 \leq \mathsf{i} \leq \ell$, $\mathsf{m_i} \geq 1$ for $1 \leq \mathsf{i} \leq \mathsf{n}$ and $\mathsf{h} \geq 0$

Logical expression $\mathsf{NER}(E_{\mathsf{def}})$
$\mathbf{ClassName}\ (\mathsf{x}) \wedge \ \mathbf{not}\ \ \mathbf{RecClassName}\ (\mathsf{x})$ $\rightarrow \exists^{\ell}_{\mathsf{i}=1}\mathsf{y_i}.\ \bigwedge^{\ell}_{\mathsf{i}=1} \mathbf{PropertyName}\ (\mathsf{x}, \mathsf{y_i}) \wedge \bigwedge^{\ell}_{\mathsf{i}=1} \left[\bigwedge^{\mathsf{k_i}}_{\mathsf{j}=1} \mathbf{NodeLabel}^{\mathsf{i}}_{\mathsf{j}}\ (\mathsf{y_i})\right] \wedge$ $\bigwedge^{\mathsf{n}}_{\mathsf{i}=1} \left[\bigwedge^{\mathsf{m_i}}_{\mathsf{j}=1} \mathbf{EdgeLabel}^{\mathsf{i}}_{\mathsf{j}}\ (\mathsf{y}_{\mathsf{S_i}}, \mathsf{y}_{\mathsf{D_i}})\right] \wedge \bigwedge^{\mathsf{h}}_{\mathsf{i}=1} \mathbf{ClassName_i}\ (\mathsf{x}) \wedge$ $\bigwedge^{\ell}_{\mathsf{i}=1} \mathbf{NewClassName}\ (\mathsf{y_i})$ $\bigwedge^{\ell}_{i=1}\ \mathbf{PropertyName}\ (\mathsf{x}, \mathsf{z_i}) \wedge \bigwedge^{\ell}_{\mathsf{i}=1} \left[\bigwedge^{\mathsf{k_i}}_{\mathsf{j}=1} \mathbf{NodeLabel}^{\mathsf{i}}_{\mathsf{j}}\ (\mathsf{z_i})\right] \wedge$ $\bigwedge^{\mathsf{n}}_{\mathsf{i}=1} \left[\bigwedge^{\mathsf{m_i}}_{\mathsf{j}=1} \mathbf{EdgeLabel}^{\mathsf{i}}_{\mathsf{j}}\ (\mathsf{z}_{\mathsf{S_i}}, \mathsf{z}_{\mathsf{D_i}})\right] \wedge \bigwedge^{\mathsf{h}}_{\mathsf{i}=1} \mathbf{ClassName_i}\ (\mathsf{x}) \wedge$ $\bigwedge^{\ell}_{\mathsf{i}=1}\ \mathbf{not}\ \ \mathbf{NewClassName}\ (\mathsf{z_i})$ $\rightarrow\ \mathbf{ClassName}\ (\mathsf{x}) \wedge\ \mathbf{RecClassName}\ (\mathsf{x})$

Table A.11: Mappings to NER for **ImpliesAndImpliedBy** expressions; we have $\ell \geq 1$, $\mathsf{n} \geq 0$, $\mathsf{k_i} \geq 0$ for $1 \leq \mathsf{i} \leq \ell$, $\mathsf{m_i} \geq 1$ for $1 \leq \mathsf{i} \leq \mathsf{n}$ and $\mathsf{h} \geq\ 0$; **RecClassName** and **NewClassName** are fresh unary predicates unique for each **ClassName**

Element $E_{\mathsf{def-inv}}$ of the BNF grammar
ClassName EquivalentTo Inverse (**PropertyName**) SOME Graph (Nodes (NI_1 $\mathbf{NodeLabel}^1_1$ ... $\mathbf{NodeLabel}^1_{\mathsf{k}_1}$, ..., NI_ℓ $\mathbf{NodeLabel}^\ell_1$ ... $\mathbf{NodeLabel}^\ell_{\mathsf{k}_\ell}$) Edges (S_1 D_1 $\mathbf{EdgeLabel}^1_1$ ... $\mathbf{EdgeLabel}^1_{\mathsf{m}_1}$, ..., S_n D_n $\mathbf{EdgeLabel}^\mathsf{n}_1$ ... $\mathbf{EdgeLabel}^\mathsf{n}_{\mathsf{m}_\mathsf{n}}$)) AND $\mathbf{ClassName}_1$ AND ... AND $\mathbf{ClassName}_\mathsf{h}$

Table A.12: **ImpliesAndImpliedBy** expression with inverse property; we have $\ell \geq 1$, $\mathsf{n} \geq 0$, $\mathsf{k_i} \geq 0$ for $1 \leq \mathsf{i} \leq \ell$, $\mathsf{m_i} \geq 1$ for $1 \leq \mathsf{i} \leq \mathsf{n}$ and $\mathsf{h} \geq 0$

Logical expression $\mathsf{NER}(E_{\mathsf{def-inv}})$
$\mathbf{ClassName}\,(\mathsf{x}) \wedge \mathbf{not}\ \mathbf{RecClassName}\,(\mathsf{x})$ $\rightarrow \exists^\ell_{\mathsf{i}=1}\mathsf{y_i}.\ \bigwedge^\ell_{\mathsf{i}=1} \mathbf{PropertyName}\,(\mathsf{y_i},\mathsf{x}) \wedge \bigwedge^\ell_{\mathsf{i}=1}\left[\bigwedge^{\mathsf{k_i}}_{\mathsf{j}=1} \mathbf{NodeLabel}^\mathsf{i}_\mathsf{j}\,(\mathsf{y_i})\right] \wedge$ $\bigwedge^\mathsf{n}_{\mathsf{i}=1}\left[\bigwedge^{\mathsf{m_i}}_{\mathsf{j}=1} \mathbf{EdgeLabel}^\mathsf{i}_\mathsf{j}\,(\mathsf{y}_{\mathsf{S_i}},\mathsf{y}_{\mathsf{D_i}})\right] \wedge \bigwedge^\mathsf{h}_{\mathsf{i}=1} \mathbf{ClassName}_\mathsf{i}\,(\mathsf{x}) \wedge$ $\bigwedge^\ell_{\mathsf{i}=1} \mathbf{NewClassName}\,(\mathsf{y_i})$ $\bigwedge^\ell_{i=1} \mathbf{PropertyName}\,(\mathsf{z_i},\mathsf{x}) \wedge \bigwedge^\ell_{\mathsf{i}=1}\left[\bigwedge^{\mathsf{k_i}}_{\mathsf{j}=1} \mathbf{NodeLabel}^\mathsf{i}_\mathsf{j}\,(\mathsf{z_i})\right] \wedge$ $\bigwedge^\mathsf{n}_{\mathsf{i}=1}\left[\bigwedge^{\mathsf{m_i}}_{\mathsf{j}=1} \mathbf{EdgeLabel}^\mathsf{i}_\mathsf{j}\,(\mathsf{z}_{\mathsf{S_i}},\mathsf{z}_{\mathsf{D_i}})\right] \wedge \bigwedge^\mathsf{h}_{\mathsf{i}=1} \mathbf{ClassName}_\mathsf{i}\,(\mathsf{x}) \wedge$ $\bigwedge^\ell_{\mathsf{i}=1} \mathbf{not}\ \mathbf{NewClassName}\,(\mathsf{z_i})$ $\rightarrow\ \mathbf{ClassName}\,(\mathsf{x}) \wedge\ \mathbf{RecClassName}\,(\mathsf{x})$

Table A.13: Mappings to NER for **ImpliesAndImpliedBy** expressions with inverse property; we have $\ell \geq 1$, $\mathsf{n} \geq 0$, $\mathsf{k_i} \geq 0$ for $1 \leq \mathsf{i} \leq \ell$, $\mathsf{m_i} \geq 1$ for $1 \leq \mathsf{i} \leq \mathsf{n}$ and $\mathsf{h} \geq 0$; where **RecClassName** and **NewClassName** are fresh unary predicates unique for each **ClassName**

Bibliography

[1] Jmol: an open-source Java viewer for chemical structures in 3D. 202

[2] LoPStER, software and datasets. http://www.cs.ox.ac.uk/isg/people/despoina.magka/tools/Thesis-LoPStER.zip. 161, 177

[3] SMARTS reference website. `http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html`. 156

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 7, 49

[5] George W. Adamson and David Bawden. Comparison of hierarchical cluster analysis techniques for automatic classification of chemical structures. *Journal of Chemical Information and Computer Sciences*, 21(4):204–209, 1981. 175

[6] Awny Alnusair, Tian Zhao, and Gongjun Yan. Automatic recognition of design motifs using semantic conditions. In Sung Y. Shin and José Carlos Maldonado, editors, *SAC*, pages 1062–1067. ACM, 2013. 9

[7] Mario Alviano, Wolfgang Faber, Nicola Leone, and Marco Manna. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *TPLP*, 12(4-5):701–718, 2012. 204

[8] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming.*, pages 89–148. Morgan Kaufmann, 1988. 60

[9] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *J. Log. Program.*, 19/20:9–71, 1994. 61, 129

[10] Eva Armengol and Enric Plaza. An ontological approach to represent molecular structure information. In José Luís Oliveira, Victor Maojo, Fernando Martín-Sánchez, and António Sousa Pereira, editors, *ISBMDA*, volume 3745 of *Lecture Notes in Computer Science*, pages 294–304. Springer, 2005. 170

[11] SS Arnon, R Schechter, T V Inglesby, D A Henderson, J G Bartlett, M S Ascher, E Eitzen, A D Fine, J Hauer, M Layton, S Lillibridge, M T Osterholm, T O'Toole, G Parker, T M Perl, P K Russell, D L Swerdlow, and K Tonat. Botulinum toxin as a biological weapon: medical and public health management. *Journal of the American Medical Association*, 285(8):1059–70, 2001. 50

[12] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, August 2007. 3, 24, 49

[13] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artif. Intell.*, 88(1-2):195–213, 1996. 165

[14] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008. 27

[15] Jean-François Baget. Improving the forward chaining algorithm for conceptual graphs rules. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *KR*, pages 407–414. AAAI Press, 2004. 95

[16] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. Extending decidable cases for rules with existential variables. In Boutilier [31], pages 677–682. 52

[17] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9–10):1620–1654, 2011. 8, 52, 57, 95, 98, 106

[18] Jean-François Baget and Marie-Laure Mugnier. Extensions of simple conceptual graphs: the complexity of rules and constraints. *J. Artif. Intell. Res. (JAIR)*, 16:425–465, 2002. 53

[19] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proc. of IJCAI*, pages 712–717, 2011. 52

[20] Jean-François Baget, Marie-Laure Mugnier, and Michaël Thomazo. Towards farsighted dependencies for existential rules. In Sebastian Rudolph and Claudio Gutierrez, editors, *RR*, volume 6902 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2011. 57, 95

[21] John M. Barnard and Geoffrey M. Downs. Clustering of chemical structures on the basis of two-dimensional similarity measures. *Journal of Chemical Information and Computer Sciences*, 32(6):644–649, 1992. 175

[22] M. Bauer, D. Brand, M. Fischer, A. Meyer, and M. Paterson. A note on disjunctive form tautologies. *ACM SIGACT News*, 5(2):17–20, 1973. 138

[23] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language reference. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-ref/`. 5

[24] Catriel Beeri and Moshe Y. Vardi. The implication problem for data dependencies. In *Proc. of the 8th Colloquium on Automata, Languages and Programming (ICALP 1981)*, pages 73–85, 1981. 49, 51

[25] Robert Berger. *Undecidability of the Domino Problem* , volume 66. Memoirs of the American Mathematical Society, 1966. 34

[26] Nicole Bidoit and Christine Froidevaux. Negation by default and unstratifiable logic programs. *Theor. Comput. Sci.*, 78(1):86–112, 1991. 102, 129, 131, 132, 135

[27] B. Bishop and S. Bojanov. Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In M. Dumontier and M. Courtot, editors, *Proc. of the OWL: Experiences and Directions Workshop (OWLED 2011)*, volume 796 of *CEUR WS Proceedings*, June 5–6 2011. 198

[28] Claudia Bobach, Timo Böhme, Ulf Laube, Anett Püschel, and Lutz Weber. Automated compound classification using a chemical ontology. *J. Cheminformatics*, 4:40, 2012. 176

[29] Christian Boelling, Michel Dumontier, Michael Weidlich, and Hermann-Georg Holzhütter. Role-based representation and inference of biochemical processes. In Ronald Cornet and Robert Stevens, editors, *ICBO*, volume 897 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. 193

[30] Pierre Bourhis, Michael Morak, and Andreas Pieris. The impact of disjunction on query answering under guarded-based existential rules. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. AAAI Press, 2013. 204

[31] Craig Boutilier, editor. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 2009. 220, 235

[32] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985. 4

[33] Martin Bravenboer and Yannis Smaragdakis. Strictly declarative specification of sophisticated points-to analyses. In Shail Arora and Gary T. Leavens, editors, *OOPSLA*, pages 243–262. ACM, 2009. 7

[34] Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012. 225, 231, 244

[35] Gerhard Brewka and Jérôme Lang, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*. AAAI Press, 2008. 223, 235, 238

[36] Alexander Brodsky and Yehoshua Sagiv. Inference of monotonicity constraints in Datalog programs. *Ann. Math. Artif. Intell.*, 26(1-4):29–57, 1999. 152

[37] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In Ian Horrocks and James A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002. 198

[38] B Buchanan, Georgia Sutherland, and EA Feigenbaum. HEURISTIC DENDRAL: a program for generating explanatory hypotheses in organic chemistry. *Machine Intelligence*, 4:209, 1969. 2

[39] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In Brewka and Lang [35], pages 70–80. 49, 52

[40] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In Paredaens and Su [195], pages 77–86. 44, 52, 56

[41] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012. 133

[42] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010. 8, 52, 53

[43] Andrea Calì, Georg Gottlob, and Andreas Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010. 52

[44] Andrea Calì, Georg Gottlob, and Andreas Pieris. Query answering under non-guarded rules in Datalog+/-. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010)*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010. 75, 77, 79, 89, 90

[45] Andrea Calì, Georg Gottlob, and Andreas Pieris. New expressive languages for ontological query answering. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011. 52

[46] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. 102

[47] F. Calimeri, S. Cozza, G. Ianni, and N. Leone. Computable functions in ASP: Theory and implementation. In M. Garcia de la Banda and E. Pontelli, editors, *Proc. of the 24th Int. Conf. on Logic Programming (ICLP 2008)*, volume 5366 of *LNCS*, pages 407–424, Udine, Italy, December 9–13 2008. Springer. 57, 59

[48] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In *LPNMR*, pages 388–403, 2011. 7

[49] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations.

In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW (Alternate Track Papers & Posters)*, pages 74–83. ACM, 2004. 198

[50] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989. 7

[51] Monica Chagoyen and Florencio Pazos. MBRole:enrichment analysis of metabolomic data. *Bioinformatics*, 27(5):730–731, 2011. 174

[52] J Chan, R Kishore, P Sternberg, and K Van Auken. The gene ontology: enhancements for 2011. *Nucleic Acids Research*, 40(D1):D559–D564, 2012. 5, 174

[53] Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded implicational dependencies and their inference problem. In *STOC*, pages 342–354. ACM, 1981. 51

[54] Vinay K. Chaudhri and Tran Cao Son. Specifying and reasoning with underspecified knowledge bases using answer set programming. In Brewka et al. [34]. 194

[55] Leonid Chepelev and Michel Dumontier. Chemical Entity Semantic Specification: Knowledge representation for efficient semantic cheminformatics and facile data integration. *Journal of Cheminformatics*, 3(20), 2011. 6

[56] Leonid Chepelev and Michel Dumontier. Semantic Web integration of Cheminformatics resources with the SADI framework. *Journal of Cheminformatics*, 3(16), 2011. 6

[57] Leonid L. Chepelev, Alexandre Riazanov, Alexandre Kouznetsov, Hong Sang Low, Michel Dumontier, and Christopher J. O. Baker. Prototype Semantic Infrastructure for Automated Small Molecule Classification and Annotation in Lipidomics. *BMC Bioinformatics*, 12:303, 2011. 174

[58] Jooyoung Choi, Melissa J. Davis, Andrew F. Newman, and Mark A. Ragan. A semantic web ontology for small molecules and their biological targets. *Journal of chemical information and modeling*, 50(5):732–741, May 2010. 6

[59] Peter Cholak and Howard A. Blair. The complexity of local stratification. *Fundam. Inform.*, 21(4):333–344, 1994. 130

[60] Ronald Cornet and Nicolette de Keizer. Forty years of SNOMED: a literature review. *BMC medical informatics and decision making*, 8(Suppl 1):S2, 2008. 5

[61] Mélanie Courtot, Nick Juty, Christian Knüpfer, Dagmar Waltemath, Anna Zhukova, Andreas Dräger, Michel Dumontier, Andrew Finney, Martin Golebiewski, Janna Hastings, Stefan Hoops, Sarah Keating, Douglas B. Kell, Samuel Kerrien, James Lawson, Allyson Lister, James Lu, Rainer Machne, Pedro Mendes, Matthew Pocock, Nicolas Rodriguez, Alice Villéger, Darren J. Wilkinson, Sarala Wimalaratne, Camille Laibe, Michael Hucka, and Nicolas Le Novère. Controlled vocabularies and semantics in systems biology. *Molecular Systems Biology*, 7(1):543, 2011. 6

[62] David Croft, Gavin O'Kelly, Guanming Wu, Robin Haw, Marc Gillespie, Lisa Matthews, Michael Caudy, Phani Garapati, Gopal Gopinath, Bijay Jassal, Steven Jupe, Irina Kalatskaya, Shahana Mahajan, Bruce May, Nelson Ndegwa, Esther Schmidt, Veronica Shamovsky, Christina Yung, Ewan Birney, Henning Hermjakob, Peter D'Eustachio, and Lincoln Stein. Reactome: a database of reactions, pathways and biological processes. *Nucleic Acids Research*, 39(Database-Issue):691–697, 2011. 174

[63] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *J. Web Sem.*, 6(4):309–322, 2008. 5, 24

[64] Arthur Dalby, James G. Nourse, W. Douglas Hounshell, Ann K. I. Gushurst, David L. Grier, Burton A. Leland, and John Laufer. Description of several

chemical structure file formats used by computer programs developed at Molecular Design Limited. *Journal of Chemical Information and Computer Sciences*, 32(3):244–255, 1992. 177

[65] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001. 7, 85, 86, 102, 129

[66] Paula de Matos, Rafael Alcántara, Adriano Dekker, Marcus Ennis, Janna Hastings, Kenneth Haug, Inmaculada Spiteri, Steve Turner, and Christoph Steinbeck. Chemical Entities of Biological Interest: an update. *Nucleic Acids Research*, 38(Database-Issue):249–254, 2010. 174

[67] D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19–20:199–260, 1994. 59

[68] Kirill Degtyarenko, Paula de Matos, Marcus Ennis, Janna Hastings, Martin Zbinden, Alan McNaught, Rafael Alcántara, Michael Darsow, Mickaël Guedj, and Michael Ashburner. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research*, 36(Database-Issue):344–350, 2008. 174

[69] Pierre Deransart, AbdelAli Ed-Dbali, and Laurent Cervoni. *Prolog - the standard: reference manual.* Springer, 1996. 6, 7

[70] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George pis. Frequent Substructure-Based Approaches for Classifying Chemical Compounds. *IEEE TKDE*, 17(8):1036–1050, 2005. 175

[71] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *Proc. of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2008)*, pages 149–158, 2008. 51, 56, 95, 98, 106, 132

[72] Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors. *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. 231, 246

[73] Michel Dumontier. Molecular Symmetry and Specialization of Atomic Connectivity by Class-based Reasoning of Chemical Structure. In *OWLED*, 2012. 176

[74] Michel Dumontier and Natalia Villanueva-Rosales. Towards pharmacogenomics knowledge discovery with the semantic web. *Briefings in Bioinformatics*, 10(2):153–163, 2009. 6

[75] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997. 60

[76] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, and Roman Schindlauer. Well-founded semantics for description logic programs in the semantic web. *ACM Trans. Comput. Log.*, 12(2):11, 2011. 133

[77] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539, 2008. 32, 43, 132

[78] Thomas Eiter, Thomas Krennwallner, Patrik Schneider, and Guohui Xiao. Uniform evaluation of nonmonotonic DL-programs. In Thomas Lukasiewicz and Attila Sali, editors, *FoIKS*, volume 7153 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2012. 43

[79] Thomas Eiter and Mantas Simkus. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Trans. Comput. Log.*, 11(2), 2010. 43

[80] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. 49, 53, 197

[81] Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. The semantic web in action. *Scientific American Magazine*, 297(6):90–97, 2007. 6

[82] Howard J Feldman, Michel Dumontier, Susan Ling, Norbert Haider, and Christopher WV Hogue. CO: A chemical ontology for identification of functional groups and semantic comparison of small molecules. *FEBS letters*, 579(21):4685–4691, 2005. 176

[83] Paolo Ferraris and Vladimir Lifschitz. Mathematical foundations of answer set programming. In Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! (1)*, pages 615–664. College Publications, 2005. 13

[84] João D. Ferreira and Francisco M. Couto. Semantic similarity for automatic classification of chemical compounds. *PLoS Computational Biology*, 6(9), 2010. 5, 174, 176

[85] Melvin Fitting. *First-order Logic and Automated Theorem Proving (Second Edition)*. Graduate texts in computer science. Springer, 1996. 49

[86] Peter Fox, Deborah L. McGuinness, Luca Cinquini, Patrick West, Jose Garcia, James L. Benedict, and Don Middleton. Ontology-supported scientific data frameworks: The virtual solar-terrestrial observatory experience. *Computers & Geosciences*, 35(4):724–738, 2009. 6

[87] Tim Furche, Georg Gottlob, Giovanni Grasso, Omer Gunes, Xiaonan Guo, Andrey Kravchenko, Giorgio Orsi, Christian Schallhart, Andrew Jon Sellers, and Cheng Wang. DIADEM: domain-centric, intelligent, automated data extraction methodology. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael

Rabinovich, and Steffen Staab, editors, *WWW (Companion Volume)*, pages 267–270. ACM, 2012. 8

[88] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 19

[89] M. Gebser, T. Schaub, and S. Thiele. GrinGo: A new grounder for answer set programming. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *Proc. of the 9th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, volume 4483 of *LNCS*, pages 266–271, Tempe, AZ, USA, May 15–17 2007. 10, 59, 105, 197

[90] 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. 131

[91] Michael Gelfond. Answer Sets, 2007. 13

[92] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988. 6, 17, 129

[93] Georgios Gkoutos, Paul Schofield, and Robert Hoehndorf. Computational tools for comparative phenomics: the role and promise of ontologies. *Mammalian Genome*, 23(9–10):669–679, 2012. 6

[94] Birte Glimm, Matthew Horridge, Bijan Parsia, and Peter F. Patel-Schneider. A syntax for rules in OWL 2. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. 66, 154

[95] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *J. Web Sem.*, 14:84–101, 2012. 5, 24

[96] Georg Gottlob, André Hernich, Clemens Kupke, and Thomas Lukasiewicz. Equality-friendly well-founded semantics and applications to description logics. In *AAAI*, 2012. 44, 134, 199

[97] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The lixto data extraction project - back and forth between theory and practice. In Catriel Beeri and Alin Deutsch, editors, *PODS*, pages 1–12. ACM, 2004. 7

[98] Erich Grädel. Description logics and guarded fragments of first order logic. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Description Logics*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998. 29

[99] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity conditions and their application to query answering in description logics. In Brewka et al. [34]. 49, 67, 106, 192

[100] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res. (JAIR)*, 47:741–808, 2013. 59, 67, 198

[101] Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, Alan Ruttenberg, and Michael Schneider. *OWL 2 Web Ontology Language: Mapping to RDF Graphs*. W3C Recommendation, 11 December 2012. Available at `http://www.w3.org/TR/owl2-mapping-to-rdf/`. 154

[102] Henson Graves. Representing product designs using a description graph extension to OWL 2. In Dolbear et al. [72]. 9

[103] S. Greco, F. Spezzano, and I. Trubitsyna. On the termination of logic programs with function symbols. In A. Dovier and V. Santos Costa, editors, *Proc. of the 8th Int. Conf. on Logic Programming (ICLP 2012)*, volume 17 of *Leibniz International Proceedings in Informatics*, pages 323–333, Budapest, Hungary, September 4–8 2012. 59, 95, 105

[104] Stephan Grimm, Boris Motik, and Chris Preist. Matching semantic service descriptions with local closed-world reasoning. In York Sure and John Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 575–589. Springer, 2006. 32

[105] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003. 8, 32

[106] David Gunning, Vinay K. Chaudhri, Peter Clark, Ken Barker, Shaw Yi Chaw, Mark Greaves, Benjamin N. Grosof, Alice Leung, David D. McDonald, Sunil Mishra, John Pacheco, Bruce W. Porter, Aaron Spaulding, Dan Tecuci, and Jing Tien. Project halo update - progress toward digital aristotle. *AI Magazine*, 31(3):33–58, 2010. 194

[107] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012. 5

[108] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic knowledge and action bases. *J. Artif. Intell. Res. (JAIR)*, 46:651–686, 2013. 197

[109] Juris Hartmanis and Richard Edwin Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, pages 285–306, 1965. 20

[110] Janna Hastings, Leonid Chepelev, Egon Willighagen, Nico Adams, Christoph Steinbeck, and Michel Dumontier. The chemical information ontology: Provenance and disambiguation for chemical data on the biological semantic web. *PLoS ONE*, 6(10), 10 2011. 6, 176

[111] Janna Hastings, Paula de Matos, Adriano Dekker, Marcus Ennis, Bhavana Harsha, Namrata Kale, Venkatesh Muthukrishnan, Gareth Owen, Steve Turner, Mark Williams, and Christoph Steinbeck. The ChEBI reference database and

ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Research*, 41(Database-Issue):456–463, 2013. 9, 141, 174, 182

[112] Janna Hastings, Michel Dumontier, Duncan Hull, Matthew Horridge, Christoph Steinbeck, Robert Stevens, Ulrike Sattler, Tertia Hörne, and Katarina Britz. Representing Chemicals Using OWL, Description Graphs and Rules. In *OWLED*, volume 614, 2010. 176, 191, 201

[113] Janna Hastings, Despoina Magka, Colin R. Batchelor, Lian Duan, Robert Stevens, Marcus Ennis, and Christoph Steinbeck. Structure-based classification and ontology in chemistry. *J. Cheminformatics*, 4:8, 2012. 175

[114] Harold E Helson. Structure diagram generation. *Reviews in Computational Chemistry, Volume 13*, pages 313–398, 2007. 157

[115] Lane A. Hemaspaandra. SIGACT news complexity theory column 36. *SIGACT News*, 33(2):34–47, 2002. 20

[116] André Hernich, Clemens Kupke, Thomas Lukasiewicz, and Georg Gottlob. Well-founded semantics for extended datalog and ontological reasoning. In Richard Hull and Wenfei Fan, editors, *PODS*, pages 225–236. ACM, 2013. 134, 199

[117] Robert Hoehndorf, Colin R. Batchelor, Thomas Bittner, Michel Dumontier, Karen Eilbeck, Rob Knight, Chris Mungall, Jane S. Richardson, Jesse Stombaugh, Eric Westhof, Craig L. Zirbel, and Neocles Leontis. The RNA ontology (RNAO): An ontology for integrating rna sequence and structure data. *Applied Ontology*, 6(1):53–89, 2011. 6

[118] Robert Hoehndorf, Michel Dumontier, and Georgios V. Gkoutos. Identifying aberrant pathways through integrated analysis of knowledge in pharmacogenomics. *Bioinformatics*, 28(16):2169–2175, 2012. 6, 174

[119] Robert Hoehndorf, Michel Dumontier, Anika Oellrich, Dietrich Rebholz-Schuhmann, Paul N. Schofield, and Georgios V. Gkoutos. Interoperability be-

tween biomedical ontologies through relation expansion, upper-level ontologies and automatic reasoning. *PLoS ONE*, 6(7), 07 2011. 6

[120] Robert Hoehndorf, Midori A. Harris, Heinrich Herre, Gabriella Rustici, and Georgios V. Gkoutos. Semantic integration of physiology phenotypes with an application to the cellular phenotype ontology. *Bioinformatics*, 28(13):1783–1789, 2012. 6

[121] Rinke Hoekstra and Joost Breuker. Polishing diamonds in OWL 2. In Aldo Gangemi and Jérôme Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 64–73. Springer, 2008. 9

[122] Matthew Horridge. *Justification based explanation in ontologies*. PhD thesis, University of Manchester, 2011. 202

[123] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester OWL syntax. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. 66, 154

[124] I. Horrocks. A comparison of two terminological knowledge representation systems. Master's thesis, University of Manchester, 1995. 4

[125] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997. 5, 49

[126] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In Anthony G. Cohn, Lenhart K. Schubert, and Stuart C. Shapiro, editors, *KR*, pages 636–649. Morgan Kaufmann, 1998. 5

[127] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 57–67. AAAI Press, 2006. 5, 24, 27

[128] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW*, pages 723–731. ACM, 2004. 32, 34

[129] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at `http://www.w3.org/Submission/SWRL/`. 8, 9, 32, 202

[130] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Sem.*, 1(1):7–26, 2003. 5, 24

[131] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. Datalog and emerging applications: an interactive tutorial. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *SIGMOD Conference*, pages 1213–1216. ACM, 2011. 7

[132] Duncan Hull. GO faster ChEBI with reasonable biochemistry. *Nature Precedings*, (713), 2008. 175

[133] Trevor Jim. SD3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society, 2001. 7

[134] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984. 49

[135] Yevgeny Kazakov. RIQ and SROIQ are harder than SHOIQ. In Brewka and Lang [35], pages 274–284. 27

[136] Yevgeny Kazakov. Consequence-driven reasoning for Horn SHIQ ontologies. In Boutilier [31], pages 2040–2045. 5, 24

[137] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. Concurrent classification of EL ontologies. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2011. 5, 24

[138] Ross D King, Stephen H Muggleton, Ashwin Srinivasan, and MJ Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996. 175

[139] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. OWLIM: A pragmatic semantic repository for OWL. In Mike Dean, Yuanbo Guo, Woochun Jun, Roland Kaschek, Shonali Krishnaswamy, Zhengxiang Pan, and Quan Z. Sheng, editors, *WISE Workshops*, pages 182–192, 2005. 198

[140] Pavel Klinov and Matthew Horridge, editors. *Proceedings of OWL: Experiences and Directions Workshop 2012, Heraklion, Crete, Greece, May 27-28, 2012*, volume 849, 2012. 240, 247

[141] Graham Klyne, Jeremy J Carroll, and Brian McBride. Resource description framework (RDF): Concepts and abstract syntax. *W3C recommendation*, 10, 2004. 202

[142] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004. 202

[143] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to query answering in dl-lite. In

Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *KR*. AAAI Press, 2010. 198

[144] Mykola Konyk, Alexander De Leon Battista, and Michel Dumontier. Chemical knowledge for the semantic web. In *DILS*, pages 169–176. Springer, 2008. 176, 191

[145] S. Krause, E. L. Willighagen, and C. Steinbeck. JChemPaint - using the collaborative forces of the internet to develop a free editor for 2D chemical structures. *Molecules*, 5:93–98, 2000. 202

[146] M. Krötzsch and S. Rudolph. On the relationship of joint acyclicity and super-weak acyclicity. Technical Report 3037, Institute AIFB, Karlsruhe Institute of Technology, 2013. Available online at `http://www.aifb.kit.edu/web/Techreport3037`. 105, 109

[147] Markus Krötzsch. *Description Logic Rules*, volume 008 of *Studies on the Semantic Web*. IOS Press/AKA, 2010. 32, 34, 44

[148] Markus Krötzsch, Despoina Magka, and Ian Horrocks. Concrete results on abstract rules. In Pedro Cabalar and Tran Cao Son, editors, *LPNMR*, volume 8148 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2013. 204

[149] Markus Krötzsch and Sebastian Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In Toby Walsh, editor, *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, pages 963–968. AAAI Press IJCAI, 2011. 52, 56, 102

[150] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012. 25

[151] Donna Kurtz, Greg Parker, David M. Shotton, Graham Klyne, Florian Schroff, Andrew Zisserman, and Yorick Wilks. CLAROS - bringing classical art to a global public. In *eScience*, pages 20–27. IEEE Computer Society, 2009. 6

[152] N. Le Novère, M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M.I. Aladjem, S.M. Wimalaratne, et al. The systems biology graphical notation. *Nature biotechnology*, 27(8):735–741, 2009. 65

[153] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006. 10, 59, 177, 197

[154] Paea Lependu, Mark A. Musen, and Nigam H. Shah. Enabling enrichment analysis with the human disease ontology. *J. of Biomedical Informatics*, 44:S31–S38, December 2011. 174

[155] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artif. Intell.*, 104(1-2):165–209, 1998. 9, 32

[156] Chen Li, Marco Donizelli, Nicolas Rodriguez, Harish Dharuri, Lukas Endler, Vijayalakshmi Chelliah, Lu Li, Enuo He, Arnaud Henry, Melanie I Stefan, et al. BioModels database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC systems biology*, 4(1):92, 2010. 174

[157] Y. Lierler and V. Lifschitz. One more decidable class of finitely ground programs. In P. M. Hill and D. Scott Warren, editors, *Proc. of the 25th Int. Conf. on Logic Programming (ICLP 2009)*, volume 5649 of *LNCS*, pages 489–493, Pasadena, CA, USA, July 14–17 2009. Springer. 58, 59, 105, 107

[158] Vladimir Lifschitz. What is answer set programming? In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1594–1597. AAAI Press, 2008. 13

[159] Fangzhen Lin and Yisong Wang. Answer set programming with functions. In Brewka and Lang [35], pages 454–465. 59

[160] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. 6

[161] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009. 7

[162] H. Low, C. Baker, A. Garcia, and M. Wenk. An OWL-DL ontology for classification of lipids. In *ICBO*, page 3, 2009. 193

[163] Thomas Lukasiewicz. A novel combination of answer set programming with description logics for the semantic web. *IEEE Trans. Knowl. Data Eng.*, 22(11):1577–1592, 2010. 133

[164] Despoina Magka. Representing graph-based structures with logic., November 2010. Transfer report. 9

[165] Despoina Magka. Ontology-based classification of chemicals: a logic programming approach. In Adrian Paschke, Albert Burger, Paolo Romano, M. Scott Marshall, and Andrea Splendiani, editors, *SWAT4LS*, volume 952 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. 178

[166] Despoina Magka, Yevgeny Kazakov, and Ian Horrocks. Tractable extensions of the description logic *el* with numerical datatypes. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR*, volume 6173 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2010. 204

[167] Despoina Magka, Yevgeny Kazakov, and Ian Horrocks. Tractable extensions of the description logic el with numerical datatypes. In Volker Haarslev, David Toman, and Grant E. Weddell, editors, *Description Logics*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. 204

[168] Despoina Magka, Yevgeny Kazakov, and Ian Horrocks. Tractable extensions of the description logic EL with numerical datatypes. *J. Autom. Reasoning*, 47(4):427–450, 2011. 204

[169] Despoina Magka, Markus Krötzsch, and Ian Horrocks. A rule-based ontological framework for the classification of molecules. *Journal of Biomedical Semantics*. accepted for publication. 178

[170] Despoina Magka, Markus Krötzsch, and Ian Horrocks. Computing stable models for nonmonotonic existential rules. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. AAAI Press, 2013. 67, 115, 135

[171] Despoina Magka, Markus Krötzsch, and Ian Horrocks. Nonmonotonic existential rules for non-tree-shaped ontological modelling. In *Proceedings of the The 26th International Workshop on Description Logics (DL 2013)*, 2013. 179

[172] Despoina Magka, Boris Motik, and Ian Horrocks. Chemical knowledge representation with description graphs and logic programming. In Adrian Paschke, Albert Burger, Paolo Romano, M. Scott Marshall, and Andrea Splendiani, editors, *SWAT4LS*, pages 74–75. ACM, 2011. 176

[173] Despoina Magka, Boris Motik, and Ian Horrocks. Classifying chemicals using description graphs and logic programming. In Klinov and Horridge [140]. 176

[174] Despoina Magka, Boris Motik, and Ian Horrocks. Modelling structured domains using description graphs and logic programming. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Description Logics*, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. 44

[175] Despoina Magka, Boris Motik, and Ian Horrocks. Modelling structured domains using description graphs and logic programming. In Simperl et al. [217], pages 330–344. 73, 192

[176] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979. 49, 95

[177] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In Paredaens and Su [195], pages 13–22. 49, 53, 78

[178] Bruno Marnette. *Tractable schema mappings under oblivious termination*. PhD thesis, University of Oxford, 2010. 56

[179] G. Meditskos and N. Bassiliades. Combining a DL reasoner and a rule engine for improving entailment-based OWL reasoning. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *Proc. of the 7th Int. Semantic Web Conf. (ISWC 2008)*, volume 5318 of *LNCS*, pages 277–292. Springer, 2008. 198

[180] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009. 56

[181] Dmitrii Mendeleev. Über die Beziehungen der Eigenschaften zu den Atomgewichten der Elemente. *Zeitschrift für Chemie*, 12:405–406, 1869. 173

[182] Nestor Milyaev, David Osumi-Sutherland, Simon Reeve, Nicholas Burton, Richard A. Baldock, and J. Douglas Armstrong. The virtual fly brain browser and query interface. *Bioinformatics*, 28(3):411–415, 2012. 6

[183] MG Moser. An overview of NIKL, the new implementation of KL-ONE. *Research in Knowledge Representation and Natural Language Understanding*, 1983. 4

[184] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Representing ontologies using description logics, description graphs, and rules. *Artif. Intell.*, 173(14):1275–1309, 2009. 9, 37, 39, 40, 176, 203

[185] Boris Motik and Riccardo Rosati. Reconciling description logics and rules. *J. ACM*, 57(5), 2010. 37, 44

[186] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005. 8, 9, 32, 34

[187] Dmitry Mouromtsev, Irina Livshits, and Maxim Kolchin. Knowledge based engineering system for structural optical design. In Hamido Fujita and Roberto

Revetria, editors, *SoMeT*, volume 246 of *Frontiers in Artificial Intelligence and Applications*, pages 254–272. IOS Press, 2012. 6

[188] Marie-Laure Mugnier. Conceptual graph rules and equivalent rules: A synthesis. In Sebastian Rudolph, Frithjof Dau, and Sergei O. Kuznetsov, editors, *ICCS*, volume 5662 of *Lecture Notes in Computer Science*, pages 23–31. Springer, 2009. 96

[189] Chris Mungall. Experiences Using Logic Programming in Bioinformatics. In *ICLP*, pages 1–21, 2009. Keynote talk. 174

[190] Natalya Fridman Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne D. Storey, Christopher G. Chute, and Mark A. Musen. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 37(Web-Server-Issue):170–173, 2009. 5

[191] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In Toby Walsh, editor, *IJCAI*, pages 1039–1044. IJCAI/AAAI, 2011. 198

[192] David Osumi-Sutherland, Simon Reeve, Christopher J. Mungall, Fabian Neuhaus, Alan Ruttenberg, Gregory S. X. E. Jefferis, and J. Douglas Armstrong. A strategy for building neuroanatomy ontologies. *Bioinformatics*, 28(9):1262–1269, 2012. 6

[193] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`. 5

[194] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. 19

[195] Jan Paredaens and Jianwen Su, editors. *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,*

PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA. ACM, 2009. 223, 240

[196] Harry E. Pence and Antony Williams. ChemSpider: An Online Chemical Information Resource. *J. Chem. Educ.*, 87(11):1123–1124, 2010. 175

[197] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008. 53

[198] Olexandr Pospishniy, Sergii Stirenko, and Ashraf Abdel-Karim Helal Abu-Ein. Using semantic technologies to improve grid resource management. In *IDAACS (1)*, pages 229–232. IEEE, 2011. 6

[199] H. Przymusinska and T. C. Przymunsinski. Weakly stratified logic programs. *Fundam. Inf.*, 13(1):51–65, March 1990. 129, 130, 131

[200] Teodor C. Przymusinski. On the declarative and procedural semantics of logic programs. *J. Autom. Reasoning*, 5(2):167–205, 1989. 129, 130

[201] Teodor C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Comput.*, 9(3/4):401–424, 1991. 60

[202] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David Scott Warren, and Juliana Freire. XSB: A system for effciently computing WFS. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *LPNMR*, volume 1265 of *Lecture Notes in Computer Science*, pages 431–441. Springer, 1997. 192

[203] AL Rector, WA Nowlan, and Andrzej Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 414. American Medical Informatics Association, 1993. 4

[204] AL Rector, JE Rogers, PE Zanstra, and Egbert Van Der Haring. OpenGALEN: open source medical terminology and tools. In *AMIA Annual Symposium Pro-*

ceedings, volume 2003, page 982. American Medical Informatics Association, 2003. 5, 203

[205] Alan L. Rector, Sean Bechhofer, Carole A. Goble, Ian Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9(2):139–171, 1997. 4

[206] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over DL-Lite ontologies. In Brewka et al. [34]. 152

[207] Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. MORe: Modular combination of OWL reasoners for ontology classification. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012. 5

[208] Riccardo Rosati. Semantic and computational advantages of the safe integration of ontologies and rules. In François Fages and Sylvain Soliman, editors, *PPSWR*, volume 3703 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005. 34

[209] Riccardo Rosati. Prexto: Query rewriting under extensional constraints in DL - Lite. In Simperl et al. [217], pages 360–374. 152

[210] Kenneth A. Ross. Modular stratification and magic sets for Datalog programs with negation. *J. ACM*, 41(6):1216–1266, 1994. 129, 131

[211] Kenneth A. Ross. A syntactic stratification condition using constraints. In *SLP*, pages 76–90, 1994. 152

[212] Cornelius Rosse and José L. V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003. 203

[213] Konstantinos F. Sagonas, Terrance Swift, and David Scott Warren. The limits of fixed-order computation. *Theor. Comput. Sci.*, 254:465–499, 2001. 129, 131

[214] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *KR*, pages 421–431. Morgan Kaufmann, 1989. 4

[215] Yi-Dong Shen, Danny De Schreye, and Dean Voets. Termination prediction for general logic programs. *TPLP*, 9(6):751–780, 2009. 59

[216] Edward H Shortliffe, Randall Davis, Stanton G Axline, Bruce G Buchanan, C Cordell Green, and Stanley N Cohen. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system. *Computers and biomedical research*, 8(4):303–320, 1975. 2

[217] Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti, editors. *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, volume 7295 of *Lecture Notes in Computer Science*. Springer, 2012. 240, 244

[218] Nicholas Sioutos, Sherri de Coronado, Margaret W. Haber, Frank W. Hartel, Wen-Ling Shaiu, and Lawrence W. Wright. NCI thesaurus: A semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics*, 40(1):30–43, 2007. 5

[219] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007. 5, 24

[220] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J Mungall, et al. The OBO foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, 2007. 174

[221] Francesca Spezzano and Sergio Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010. 56

[222] Ola Spjuth, Jonathan Alvarsson, Arvid Berg, Martin Eklund, Stefan Kuhn, Carl Mäsak, Gilleain M. Torrance, Johannes Wagener, Egon L. Willighagen, Christoph Steinbeck, and Jarl E. S. Wikberg. Bioclipse 2: A scriptable integration platform for the life sciences. *BMC Bioinf.*, 10:397, 2009. 202

[223] Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artif. Intell.*, 85(1-2):277–299, 1996. 175

[224] Giorgos B. Stamou, Jacco van Ossenbruggen, Jeff Z. Pan, and Guus Schreiber. Multimedia annotations on the semantic web. *IEEE MultiMedia*, 13(1):86–90, 2006. 6

[225] Christoph Steinbeck, Christian Hoppe, Stefan Kuhn, Matteo Floris, Rajarshi Guha, and Egon L Willighagen. Recent developments of the chemistry development kit (CDK) - an open-source java library for chemo- and bioinformatics. *Curr. Pharm. Des.*, 12(17):2111–20, 2006. 177

[226] Markus Stocker and Michael Smith. Owlgres: A scalable OWL reasoner. In Dolbear et al. [72]. 5

[227] T. Syrjänen and I. Niemelä. The Smodels system. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proc. of the 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *LNAI*, pages 434–438, Vienna, Austria, September 17–19 2001. Springer. 59, 197

[228] Tommi Syrjänen. Omega-restricted logic programs. In *LPNMR*, pages 267–279, 2001. 59, 105

[229] Stephans Tobies. *Complexity results and practical algorithms for logics in knowledge representation.* PhD thesis, University of Manchester, 2001. 5

[230] N. Trinajstic. *Chemical graph theory*, volume 2. CRC press Boca Raton, FL, 1992. 65

[231] Dmitry Tsarkov and Ian Horrocks. Description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer, 2006. 5, 24

[232] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC*, pages 137–146. ACM, 1982. 20

[233] Moshe Y. Vardi. Why is modal logic so robustly decidable? In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1996. 8, 29

[234] Natalia Villanueva-Rosales and Michel Dumontier. Describing chemical functional groups in OWL-DL for the classification of chemical compounds. In *OWLED*, 2007. 42, 176, 191

[235] Nicola Vitucci, Mario Arrigoni Neri, Roberto Tedesco, and Giuseppina Gini. Semanticizing syntactic patterns in NLP processing using SPARQL-DL queries. In Klinov and Horridge [140]. 6

[236] Yanli Wang, Jewen Xiao, Tugba O. Suzek, Jian Zhang, Jiyao Wang, Zhigang Zhou, Lianyi Han, Karen Karapetyan, Svetlana Dracheva, Benjamin A. Shoemaker, Evan Bolton, Asta Gindulyte, and Stephen H. Bryant. PubChem's BioAssay Database. *Nucleic Acids Research*, 40(Database-Issue):400–412, 2012. 175

[237] Jörg K. Wegner, Aaron Sterling, Rajarshi Guha, Andreas Bender, Jean-Loup Faulon, Janna Hastings, Noel M. O'Boyle, John P. Overington, Herman van Vlijmen, and Egon L. Willighagen. Cheminformatics. *Commun. ACM*, 55(11):65–75, 2012. 175

[238] David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. 156

[239] David Weininger. SMILES, 3. depict. graphical depiction of chemical structures. *Journal of Chemical Information and Computer Sciences*, 30(3):237–243, 1990. 156

[240] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES. 2. algorithm for generation of unique smiles notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101, 1989. 156

[241] Michael Wessel, Marko Luther, and Ralf Möller. What happened to bob? semantic data mining of context histories. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. 9

[242] David J. Wild, Ying Ding, Amit Sheth, Lee Harland, Eric M. Gifford, and Michael S. Lajiness. Systems chemical biology and the semantic web: what they mean for the future of drug discovery research. *Drug Discovery Today*, 2012. 6

[243] Egon Willighagen and Martin Brändle. Resource description framework technologies in chemistry. *Journal of Cheminformatics*, 3(1):15, 2011. 6

[244] Sarala M. Wimalaratne, Pierre Grenon, Robert Hoehndorf, Georgios V. Gkoutos, and Bernard de Bono. An infrastructure for ontology-based information systems in biomedicine: RICORDO case study. *Bioinformatics*, 28(3):448–450, 2012. 6

[245] K. J. Wolstencroft, R. Stevens, L. Tabernero, and A. Brass. Phosphabase: an ontology-driven database resource for protein phosphatases. *Proteins*, 58(2):290–4, 2005. 6

[246] Katy Wolstencroft, Andy Brass, Ian Horrocks, Phillip W. Lord, Ulrike Sattler, Daniele Turi, and Robert Stevens. A Little Semantic Web Goes a Long Way in Biology. In *ISWC*, 2005. 174

[247] Katy Wolstencroft, Phillip W. Lord, Lydia Tabernero, Andy Brass, and Robert Stevens. Protein classification using ontology classification. In *ISMB (Supplement of Bioinformatics)*, pages 530–538, 2006. 6

[248] William A Woods and James G Schmolze. The KL-ONE family. *Computers & Mathematics with Applications*, 23(2):133–177, 1992. 4

[249] Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, and Jagannathan Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In *Proc. of the 2008 IEEE 24th Int. Conf. on Data Engineering (ICDE 2008)*, pages 1239–1248. IEEE Computer Society, 2008. 198

[250] Qian Zhu, Yuyin Sun, Sashikiran Challa, Ying Ding, Michael S. Lajiness, and David J. Wild. Semantic inference using chemogenomics data for drug discovery. *BMC Bioinformatics*, 12:256, 2011. 6